IBM Field Engineering

Theory of Operation

7201-02 Computing Element

PREFACE

This manual describes the operation of the IBM 7201-02 Computing Element which is used in the IBM 9020D and 9020E systems for air traffic control. The manual is divided into four chapters and four appendices:

Chapter 1, Introduction, describes how the Computing Element fits into the 9020D and 9020E Systems, discusses the program execution and control, introduces the functional units (adders, registers, counters, etc.) in the Computing Element, and describes the instruction set with which the Computing Element operates.

Chapter 2, Functional Units, analyzes the operation of each functional unit individually, except where two functional units work together to perform a specific function (for example, variable-field-length register and its associated byte counter).

Chapter 3, Principles of Operation, analyzes in detail instruction fetching and instruction execution by instruction class.

Chapter 4, Manual Controls and Maintenance Facilities, discusses the operation of controls on the Computing Element's control panel and describes the maintenance facilities for the Computing Element.

Appendix A, Special Circuits, identifies the special circuits in the Computing Element.

Appendix B, Interface Lines, defines all interface lines between the CE and other elements/units in the system.

Appendix C, 1052 Adapter, describes the operation of the 1052 Adapter attached to the Computing Element.

Appendix D, Numbering Systems, Instruction Coding, and Data Formats, discusses (1) the hexadecimal (hex) number system, (2) the eight-bit zoned character codes, (3) the instruction formats and operand designations, and (4) the various data formats.

The illustrations supporting the text in this manual are divided into two groups: (1) purely instructional and (2) maintenance oriented. The purely instructional illustrations are referred to as "figures" (e.g., Figure 3-1) and are located in this manual. The maintenance-oriented illustrations are referred to as "diagrams" (e.g., Diagram 4-3, FEMDM) and are located in the companion FE Maintenance Diagrams Manual.

Prerequisite and companion manuals are:

Prerequisite Manuals

9020E System Introduction, Theory of Operation Manual, Form SFN-0103

9020D System Introduction, Theory of Operation Manual, Form SFN-0104

Companion Manuals

7201-02 Computing Element, Maintenance Diagrams Manual, Form SFN-0202

7201-02 Computing Element, Maintenance Manual, Form SFN-0203

7201-02 Computing Element, Installation Manual, Form SFN-0204

7201-02 Parts Catalog, Form SFN-0205

9020 D/E Power Controls and Distribution, Theory of Operation Manual, Form SFN-0105

First Edition (July, 1970)

This manual has been prepared by IBM Product Publications, Kingston, N.Y.

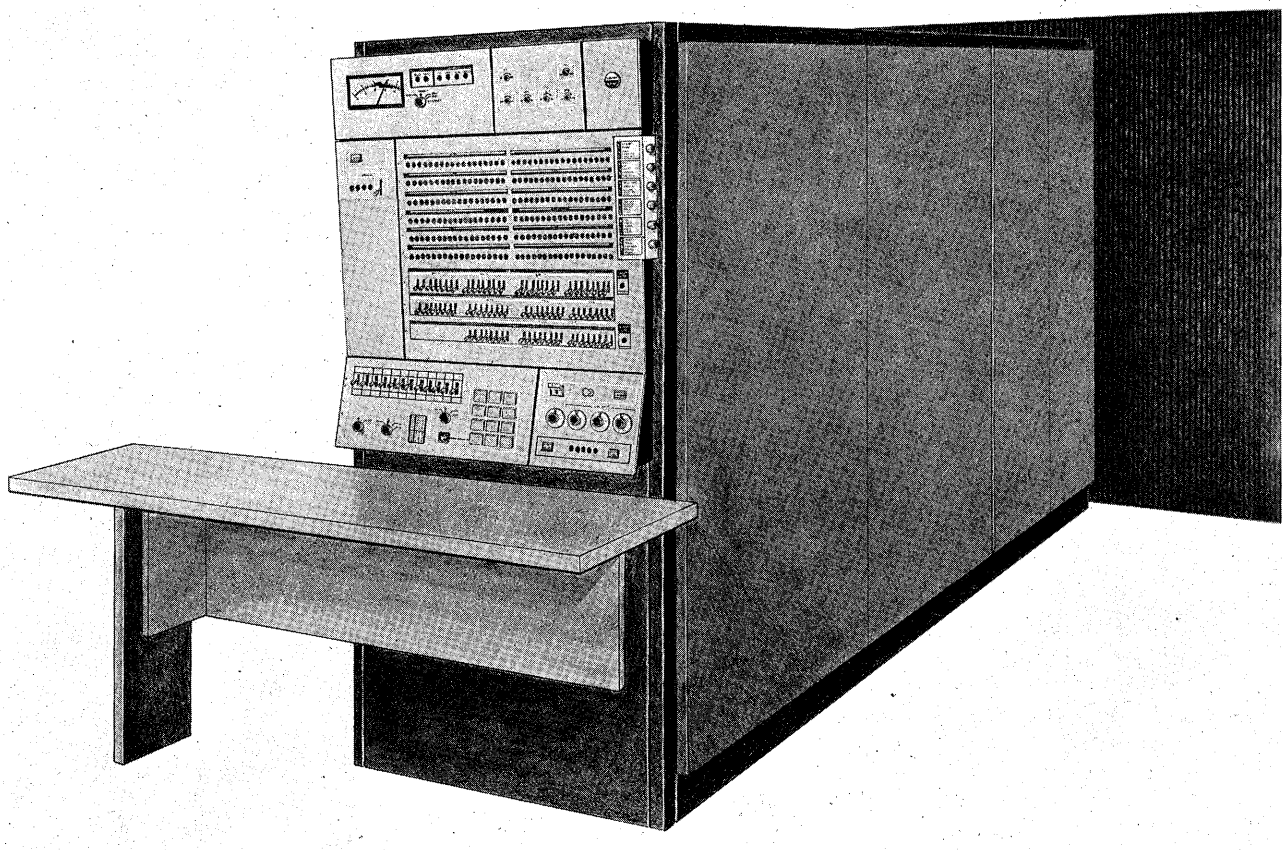
ILLUSTRATIONS

TABLES

ABBREVIATIONS

ABC	AB register byte counter
ac	alternating current
ACR	Automatic Carrier Return
adr	address, addressed, addressing
ALD	automated logic diagram
ALTN	Alternate
amp	ampere
APSA	alternate preferential storage area
ASC	address store compare
ATC	air traffic control
ATN	alternate test number
ATR	address translation register
Attn	attention
Aux	Auxiliary Magnet
BCD	binary-coded decimal
BCU	bus control unit (alternate terminology for SCI)
BL	blink
BR	brightness
BSM	basic storage module
C	capacitor
CAS	control automation system
CAW	channel address word
CB	circuit breaker
CC	condition code, also Configuration Console
CCC	Central Computer Complex
CCR	configuration control register
CCW	channel command word
CE	Computing Element
Charistic	Characteristic
CLD	control automation system logic diagram
CLU	Common Logic Unit
Cmd	command
CPU	Central Processing Unit (alternate terminology for CE)
CR	diode or Carrier Return
CROS	capacitive read-only storage
CSW	channel status word
CT	conditional terminate
CTC	channel-to-channel
CU	Control Unit
CVG	Character Vector Generator
DA	dash
DAR	diagnose accessible register
DARM	diagnose accessible register mask
DASF	Direct Access Storage Facility
DAU	Data Adapter Unit
dc	direct current
DCP	Display Channel Processor
DE	Display Element
dec	decimal
dec div	decimal divide
dec ovflo	decimal overflow
DG	Display Generator
Disc	disconnect
dly	delay
Dply	display
dsbl	disable
DSU	Disk Storage Unit
DX	first byte in a series of destination bytes
DX+1	second byte in a series of destination bytes
DX+2	third byte in a series of destination bytes
ELC	element check
end op	end operation
EOB	end of block
EOL	End-of-Line
EPO	emergency power off
ERSLT	expected result
EXC	Executive Control Program
exp ovflo	exponent overflow
exp unflo	exponent underflow
F	fuse
FEMDM	Field Engineering Maintenance Diagrams Manual
FEMI	Field Engineering Manual of Instruction
FEMM	Field Engineering Maintenance Manual
FETOM	Field Engineering Theory of Operation Manual
fix-pt ovflo	fixed-point overflow
FLT	fault locating test
flt-pt div	floating-point divide
FMTN	Format New
FMTO	Format Old
FMTW	Format Weather
FPR	Floating-point register
fract	fraction
GIS	general initialization sequence
GPR	general-purpose register
hex	hexadecimal
Hz	Hertz
IC	instruction counter
ICR	inhibit carrier return
IDES	inhibit display element stop
I-Fetch	instruction fetching
ILC	instruction length code
ILOS	inhibit logout stop
Init	initial
I/O	input/output
IOCE	Input/Output Control Element
IPL	initial program load
K	kilo; also relay
kHz	kilohertz
LAB	logical address bus
LADS	Logic Automation Documentation System
LAL	local storage address latches
LAR	local storage address register
LC	lower case
LF	line feed
LOS	logout stop
LS	local store
LSWR	local storage working register

MACH	maintenance and channel (storage)
max	maximum
MC	machine check
MCW	maintenance control word
mHz	megahertz
MMSC	maintenance mode stop clock
Mple	Multiple
MPR	multiplier
MPX	multiplex
ms	millisecond
NDT	new descriptor tables
no op	no operation
NRM	new refresh memory
NRMA	new refresh memory address
ns	nanosecond
OBS	on battery signal
ODT	old descriptor tables
op code	operation code
oper	operation
opr	operand
ORM	old refresh memory
ORMA	old refresh memory address
OTC	out of tolerance check
P	parity
PAA	parallel adder A-side
PAB	parallel adder B-side
PAL	parallel adder latch
PB	pushbutton
pf	picofarad
PK	power contactor
PP	partial product
PQ	partial quotient
priv oper	privileged operation
proc	process
prog	program
PROSAR A	previous read-only storage address register A
PROSAR B	previous read-only storage address register B
prot	protection
PS	power supply
PSA	preferential storage address
PSBA	preferential storage base address
PSBAR	preferential storage base address register
PSW	program status word
PVD	Plan View Display
R	resistor
RCU	Reconfiguration Control Unit
reg	register
RKM	Radar Keyboard Multiplexor
ROS	read-only storage
ROSAR	read-only storage address register
ROSBR	read-only storage backup register
ROSDR	read-only storage data register
RST	Reset
SAA	serial adder A-side
SAB	storage address bus, also serial adder B-side
SAL	serial adder latch
SATR	set Address Translation Register
SBA	serial adder bus A
SBB	serial adder bus B
SC	System Console
SCI	storage control interface
SCON	set Configuration Control Register
SCOPEX	scoping index
SCR	silicon-controlled rectifier
SCU	storage control unit
SDBI	storage data bus in
SDBO	storage data bus out
SE	Storage Element
Sel	select
Serv	service
signif	significance
SLT	solid logic technology
SMMC	system maintenance monitor console
SMS	standard modular system
SOROS	scan out read-only storage
spec	specification
SRL	Systems Reference Library
SSU	storage switching unit
STAT	status trigger
STC	ST register byte counter
stg	storage
SU	switch unit
sync	synchronizing
T	transformer
TC	time clock (interval timer)
TCU	tape control unit
T(DX)	table byte specified by DX
T(DX+1)	table byte specified by DX+1
TIC	transfer in channel
TN	test number
T/R	tilt/rotate
TU	tape unit
uc	upper case
uf	microfarad
usec	microsecond
UT	unconditional terminate
V	volt
VFL	variable-field length
VFR	visual flight rules
Xlat	translate
$\geq$	greater than or equal to
$\leq$	less than or equal to
=	equal to
$\neq$	not equal to
&	and



IBM 7201-02 Computing Element

CHAPTER 1. INTRODUCTION

This chapter introduces the IBM 7201-02 Computing Element (CE), which is an integral part of the IBM 9020D and 9020E systems for air traffic control. The chapter is divided into two sections.

Section 1 describes the relationship of the CE to the 9020D and 9020E systems and discusses the overall operation of the systems under the control program.

Section 2 introduces the functional units within the CE and describes the major CE operations and facilities; i.e., configuration control, storage addressing, instruction fetching and execution, interruption and direct control facilities, interfacing with other elements, and maintenance features.

SECTION 1. RELATIONSHIP OF THE CE TO THE 9020D AND 9020E SYSTEMS

The IBM 7201-02 Computing Element (CE) is the nucleus of the 9020D Central Computer Complex (CCC) and the 9020E Display Channel Processor (DCP) systems. It is the primary element for computation, logic, and control within these systems.

Because the two systems differ in structure and in the tasks they perform, the function of the CE is somewhat different in the two systems. The CEs are physically the same, however; each is equipped to perform all of the functions required by either system. Specific physical adaptation required for a CE to operate in a particular system is provided by plug cards contained in the logic. The physical plugging of these cards provides such information to the CE as type of system, amount of available storage, number of other elements installed on the system, etc. These plug cards can be altered by maintenance personnel if the system structure is changed for any reason.

The two systems are described briefly in the following paragraphs. For an extensive introduction to either system, refer to the applicable introduction manual:

9020E System Introduction Manual,
9020D System Introduction Manual,

IBM 9020D CENTRAL COMPUTER COMPLEX (CCC)

The IBM 9020D Central Computer Complex (CCC) is the collection of computers and computer-controlled equipment responsible for the real-time processing of air traffic data for subsequent use by flight controllers in performing the air traffic control task. The CCC receives data from many sources, processes it, and makes it available in the form of flight progress strips, various operational messages, and display data. It is directly responsible for outputting flight strips and operational messages, and passes partially processed radar and weather data for additional processing to an IBM 9020E Display Channel Processor (or similar display processing equipment).

A CCC may contain as many as four CEs. These CEs provide a pool of processing capability more than equal to the air traffic control (ATC) task; thus, a malfunction in one CE can be tolerated even under the heaviest workload.

In the CCC, The CEs operate in conjunction with the following elements:

1. IBM 7251-09 Storage Elements (SEs)
2. IBM 7231-02 I/O Control Elements (IOCEs)
3. IBM 7289-02 Peripheral Adapter Modules (PAMs)
4. IBM 7265-02 System Console (SC)
5. IBM 2803-01 Tape Control Units (TCUs)
6. IBM 2314-A1 Storage Control Units (SCUs)

The CEs do not contain main storage or channels; instead they depend on SEs and IOCEs to provide these functions. A maximum of ten SEs provide main storage in blocks of 524,288 bytes per SE. I/O channels are provided by IOCEs which, under control of the CEs, perform all I/O operations for the CCC. Because the IOCEs have direct access to main storage, this arrangement frees the CEs from the time-consuming process of transferring large amounts of I/O data to and from storage. A maximum of three IOCEs may be installed, each of which may have one multiplexer channel and three selector channels (except for the third IOCE, which may have only two selector channels). A selector channel operates with one device at a time; a multiplexer channel is time-shared between several devices. In addition to channels, IOCEs also contain an IOCE-processor feature, which enables a CE to delegate minor tasks to the IOCE.

The PAMs contain adapters of various types so that the CCC may interface with a variety of external equipment for data input and output. A maximum of three PAMs may be installed on the CCC system.

Up to three TCUs control the operation of IBM 2401 Magnetic Tape Units (TUs). Each TCU has a dual interface, which permits it to be connected to selector channels in two different IOCEs.

Up to three SCUs can be attached to each CE SCON interface. Each SCU has a dual interface which permits it to be connected to selector channels in two different IOCEs.

The SC provides a central location for monitoring and control of the 9020D system. Only one SC is provided, since all critical controls and indicators are duplicated elsewhere in the system. The SC contains an adapter for an IBM 1052 I/O Writer and manual controls for attaching the 1052 to any installed IOCE. An IBM 2821 I/O Control Unit is also manually switched at the SC to any of the attached IOCEs. The 2821 is the control unit for an IBM 2540 Card Reader/Punch and an IBM 1403 High Speed Printer. An interface is also provided to a system maintenance monitor console (SMMC) which monitors both the CCC and the DCP systems.

IBM 9020E DISPLAY CHANNEL PROCESSOR (DCP)

The IBM 9020E Display Channel Processor (DCP) receives radar and weather data from the CCC system via channel-to-channel (CTC) adapters, which connects IOCEs in the two systems. This data is further processed and passed on to display equipment. The DCP also handles input and output data for radar keyboard equipment. The display and rader keyboard equipment are external to the DCP system.

A DCP system may contain as many as four CEs, which, as in the CCC system, provide sufficient redundancy so that a malfunctioning CE can be tolerated under peak workload conditions. CEs in the DCP system utilize special display instructions for the processing of radar and weather data. These display instructions are implemented in the CEs of the CCC system as well, since all 7201-02 CEs are alike. However, it is only in the DCP system that these instructions are useful.

In the DCP system, the CEs operate in conjunction with the following elements:

1. IBM 7251-09 Storage Elements (SEs)
2. IBM 7289-04 Display Elements (DEs)
3. IBM 7231-02 I/O Control Elements (IOCEs)
4. IBM 7265-03 Configuration Console (CC)
5. IBM 2803-01 Tape Control Units (TCUs)
6. IBM 2701-01 Data Adapter Units (DAUs)

The SEs, IOCEs, and TCUs are the same as those in the 9020D CCC system and perform the same functions. The DEs, CC, and DAUs are used in the DCP system only.

A DE is similar to an SE in many respects. The storage section of a DE is used to buffer display data for the display equipment. The switch unit portion of a DE interfaces with up to four display generators (DGs), each of which control six character vector generators (CVGs). The CVGs request and receive data from the DEs and use the data to control plan view displays (PVDs), one PVD per CVG.

Each DE is also capable of interfacing with four CEs and receives data from them as a result of the CEs execution of the display instructions. The DEs are also provided with a wrap bus, which makes possible the testing of DEs by returning data (which would normally go to CVGs) to the CEs for checking. The CE is provided with a DE wrap interface for this purpose.

The DAU connects to radar keyboard multiplexers (RKMs) to provide a data path between IOCE multiplexer channels and keyboard equipment located at the radar consoles.

The CC is similar to the SC in the CCC system, with the following exceptions:

1. Additional units, called reconfiguration control units (RCUs), are housed within the CC. The function and purpose of the RCUs is explained in the 9020E System Introduction Manual.
2. No 1052 adapter is housed in the CC; adapters housed in the CEs are used instead.
3. The SMMC interface is of a different type. The SMMC interface for each system is explained in Chapter 3 of the respective System Introduction Manuals.
4. Additional indicators are installed to permit the status of display equipment to be indicated.

CE INTERFACING

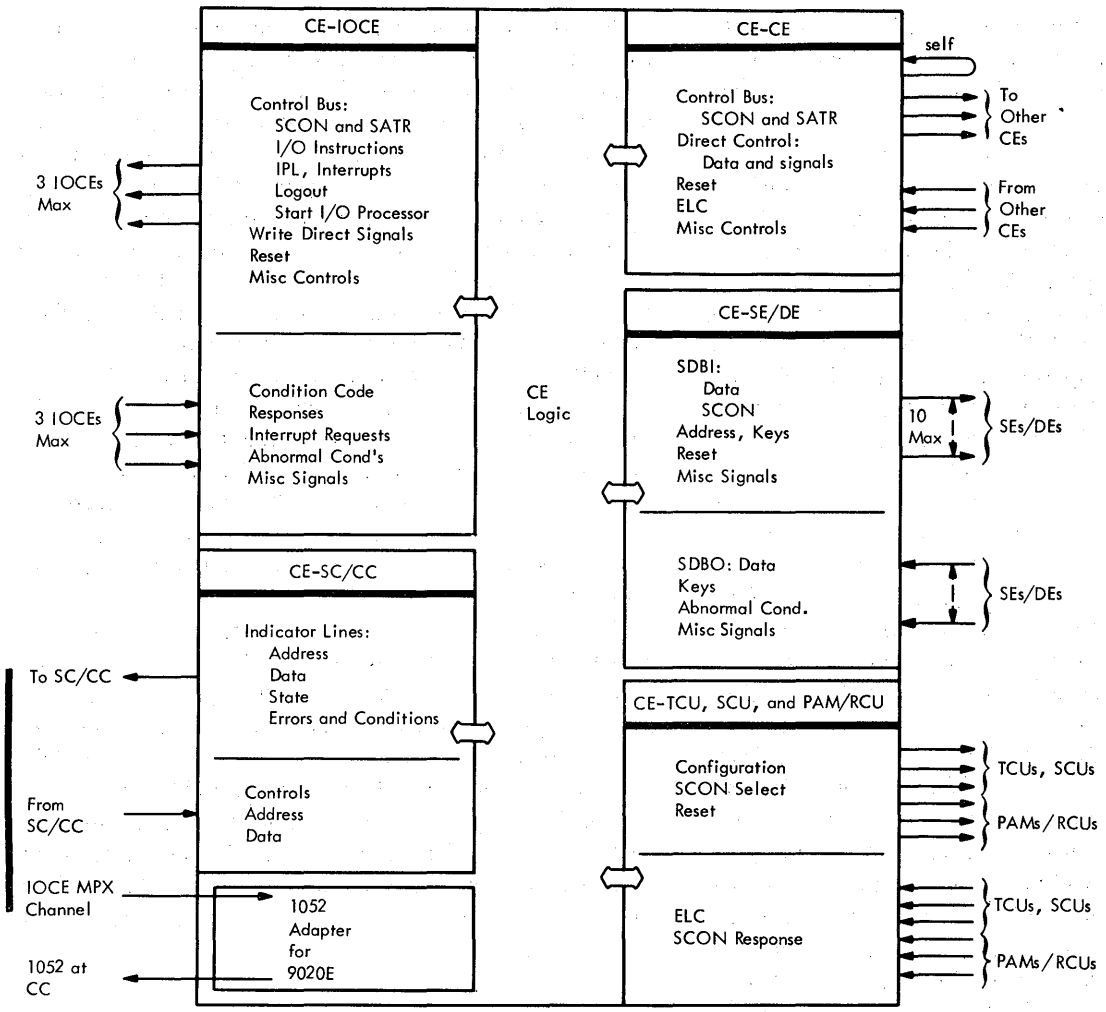
As the controlling element in the system, the CE interfaces with every major element and unit. A brief discussion on CE interfacing is presented below. (For a complete discussion of element interfacing, refer to Appendix B of this manual.)

Figure 1-1 shows CE interfacing in simplified form. Five main types of interfaces are shown:

1. CE–CE
2. CE–SE and DE
3. CE–TCU, SCU, and PAM or RCU
4. CE–IOCE
5. CE–SC or CC

The CE to CE interface provides data paths for configuration information, ATR information, and direct control. (Note that each CE also interfaces with itself for these purposes.) Other interface lines conduct the system reset, direct control signals, and the 'element check' (ELC) signal. These control lines enable any CE to monitor the operation of the other CEs and to coordinate them in multisystem operation.

Each CE interfaces with each SE and DE in the system. These interfaces are very similar. A Storage Data Bus In (SDBI) and a Storage Data Bus Out (SDBO) provide for the



●Figure 1-1. Computing Element Interfacing

transfer of data, a doubleword at a time, between the CE and storage. The SDBI is also used to transfer configuration data to SEs and DEs. No separate configuration bus exists to these elements. Separate interface lines are provided for storage addresses and keys. A number of additional interface lines permit synchronized operation and permit the CE to monitor for SE or DE errors or abnormal conditions.

In the CE to DE interface, special lines permit the testing of DEs without interfering with display equipment. Request lines from the CE to the DE permit the CE to request data from the DEs which normally would be transferred to the display equipment. A special "DE Wrap" bus is provided for return of the data to the CE.

The CE interfaces with TCUs, SCUs, and with either PAMs or RCUs, depending on whether the system is a 9020D or a 9020E. These interfaces are very similar. Provision is made for configuration data to be transferred to these units and for the signals necessary to permit the CE to select the units for configuration and monitor them for errors or abnormal conditions.

The CE to IOCE interface is relatively complex compared with the other interfaces although no direct data path is available between these two element types. Shared storage is used for this purpose. The major portion of the CE to IOCE interface consists of a control bus used for the following purposes: SCON and SATR information, IOCE-processor start, I/O instructions, IPL, interrupts, and logout. In all but the first two of these, the CE's PSBAR is placed on the bus to enable the IOCE to access the PSA. The IOCE to CE interface lines consist mainly of signal lines which permit CE and IOCE operation to be synchronized when necessary and which permit the CE to monitor for errors and abnormal conditions. Direct control signals are provided between CEs and IOCEs, but no direct control data path exists.

The CE interfaces with the SC or CC to enable the CE status to be displayed at the console and to enable various CE control panel functions to be controlled remotely from the console.

In all the interfaces mentioned, certain common characteristics exist. Provision is made for the CE to configure all elements and to monitor them for abnormal conditions. A system reset signal is also provided to each element.

Note in Figure 1-1 that lines are shown entering and leaving the CE to connect with a 1052 adapter. This adapter is housed in the CE for use in the 9020E system. It connects an IOCE multiplexer channel to a 1052 I/O writer located at the CC. (The operation of the 1052 adapter is described in Appendix C of this manual.)

CONTROL PROGRAM

Because of the size and internal processing speed of the 9020 systems, it is not practical to manually load and control the many subprograms (which run in as many as four CEs), nor is it possible to respond manually to all of the exceptional and abnormal conditions that arise and still retain real-time processing capability. Therefore, the 9020 systems are designed to operate under a control program (sometimes called a supervisor or monitor program). The control program, operating at machine speed, can respond rapidly to changes in workload and to various exceptional and abnormal conditions. This rapid response is necessary for the fail-safe, fail-soft operation required by the ATC task. The control program for the ATC system is called the executive control program (EXC).

The EXC program is responsible for many different aspects of system operation. This section discusses these various aspects and describes how certain hardware-implemented facilities of the 9020 systems enable the EXC program to achieve control over them.

Privileged Instructions

The EXC program has available to it certain privileged instructions. These instructions, which can be executed only by the control program, enable the EXC program to initialize and control a number of operations and system conditions. For example, all I/O instructions are privileged, thus placing all I/O operations under direct control of the EXC program.

The manner in which privileged instructions are reserved for control program use is shown later in this section.

Configuration Control

As noted in the system descriptions, the 9020 systems contain more elements than are required to perform the ATC task, even under peak workload conditions. The spare elements are termed redundant; they may be used to replace failing elements. Elaborate interfaces between elements enable one element to be substituted for another by changing the interface gating. This gating is controlled by a configuration control register (CCR) in each major element and unit. Because manual reconfiguration is too time-consuming, the 9020 systems are designed to be reconfigured under program control. A privileged instruction called Set Configuration (SCON) enables the EXC program to set the CCRs in each element unit.

Fields in the CCR determine for each element (1) other elements/units with which it may communicate, (2) CEs from which it may accept future reconfigurations, and (3) the element state it is to assume. The first of these is self-explanatory. The second field mentioned, called the SCON field, enables the EXC to force an element to accept reconfigurations only from the CE in which the EXC, itself, is running.

The element state field determines, for each element, which manual controls are enabled. This enables the EXC program to place the element under tight control or to permit varying degrees of manual control for such purposes as maintenance or program debugging.

Element states are discussed more fully later in this section. The subject of configuration control is discussed in relation to the CE in Section 2. System aspects of configuration control are described in detail in Chapter 4 of the system introduction manuals cited at the beginning of this chapter.

Address Translation

Storage elements are reconfigurable even though each SE represents a fixed block of addresses which ordinarily could not be substituted for a different block of fixed addresses. This substitution is made possible by an address translation facility implemented in the CEs and IOCEs. The address translation facility enables automatic translation of logical addresses from the program into physical addresses in the SE actually occupying a particular address block. This is accomplished via an address translation register (ATR) in each CE and IOCE. There are ten slots in the ATR for SE identifiers (SE and DE identifiers in the 9020E system). High-order bits of a logical address decode ATR slots rather than SEs. The EXC program can set SE identifiers into the ATR slots via another privileged instruction called set address translator (SATR). This must be done prior to executing the SCON instruction so that CEs and IOCEs will decode the proper SEs to which they are configured. By changing the ATR setting, the EXC program can substitute one SE for another in the event of an SE failure.

The subject of storage addressing is discussed more fully in Section 2 of this chapter and is described in detail in

Chapter 5 of the system introduction manuals cited at the beginning of this chapter.

Preferential Storage Areas

Each CE in the system must have a unique area of storage, called the preferential storage area (PSA), which is set aside for storing certain data critical to the operation of that CE's programs. The PSA occupies 4096 bytes of storage. As will be seen later, in the discussion of interruptions, the PSA must be accessible both to the CE's programs and to hardware.

In the 9020 systems, in which SEs may be substituted for each other, the PSA for a particular CE may be located anywhere in storage on 4096-byte boundaries. A special register, called the preferential storage base address register (PSBAR) is used in each CE and IOCE to retain the current location of the PSA. When a program running in the CE addresses the lowest 4096 bytes of storage available to that CE, address bits from PSBAR are substituted automatically so that the actual location of the PSA is generated. Low-order bits of the original address are retained so that the program may specify particular locations within the 4096-byte PSA.

Again, a privileged instruction is available to the EXC program for loading the PSBAR. This instruction is called Load Preferential Storage Base Address (LPSB).

Provision is made to automatically search the ATR for the next configured SE should a CE fail to gain access to the SE containing its PSA. The EXC program is responsible for establishing, in addition to PSAs, alternate PSAs for each CE. The alternate PSA must be kept updated so that data critical to continued operation will be available in the event of a malfunction in the PSA SE.

This subject is discussed more fully in Section 2 of this chapter and is described in detail in Chapter 5 of the system introduction manuals cited at the beginning of this chapter.

Direct Control

Provision has been made for CEs to pass data directly between them, one byte at a time, and for direct control signals to pass between them. Certain direct control signals may be passed between CEs and IOCEs also, but no direct control data path exists between them. This facility provides the EXC program with the ability to coordinate simultaneous operation of CEs so that more than one subsystem may operate under EXC program control. Via this facility, the EXC program can start and stop CEs and IOCE-processors and direct their activities into a coordinated multisystem operation.

The direct control instructions are discussed in Section 2 of this chapter. A description of multisystem operation is given in Chapter 6 of the system introduction manuals cited at the beginning of this chapter.

Interruptions

An interruption may be considered as an automatic program branch. That is, the current sequence of instructions is interrupted, and a branch (interruption) is taken to a new instruction sequence. The interruption facility of the 9020 is hardware-implemented and, in most cases, is completely automatic.

When an interruption occurs, the current status of the CE is stored in a fixed location within the CE's PSA. Information is fetched from another location in the PSA to control CE operation while the interruption is being handled. For this reason, each CE must have a unique PSA accessible by hardware as well as the program.

The status of the CE is saved at the time of the interruption in a doubleword called a program status word (PSW). The contents of the PSW describe the CE's status completely so that the original instruction sequence can be continued later as though no interruption had occurred. PSWs in which status is saved during an interruption are called old PSWs. Information fetched to control the CE during the handling of an interruption is in the form of a new PSW. Both old and new PSWs have fixed locations within a CE's PSA so that they can be stored and fetched automatically when an interruption occurs.

The EXC program is responsible for establishing the new PSWs in its own PSA so that they will properly direct the CE's operation when called into use during an interruption. This includes setting next-instruction addresses in each new PSW that point to instructions that will handle the particular type of interruption.

Five classes of interruptions occur in the 9020 systems:

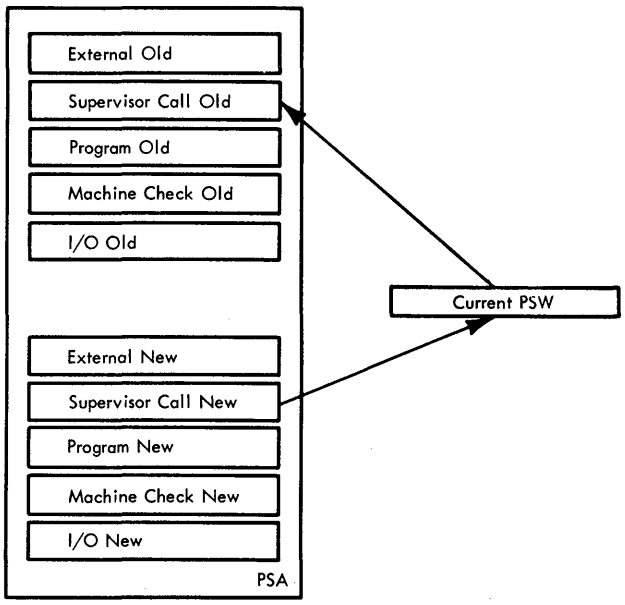
1. Machine-check
2. Program
3. Supervisor call
4. External
5. Input/output

Both an old and a new PSW location exist in a CE's PSA for each class of interruption; thus, there are five old PSW locations and five new PSW locations. The current status of the CE is contained in the current PSW, which is not located in storage at all. The current PSW is the total status of the CE as represented by a number of control triggers in the CE circuitry. There is no single location for all the bits of the current PSW.

As an interruption occurs, the condition of all the triggers making up the current PSW is saved in the old PSW for that interruption class. The new PSW for that interruption class is then fetched, and the triggers making up the current PSW are set to agree with the bits of the new PSW. This is shown in Figure 1-1 for a supervisor call interruption.

A special field in the old PSW is used to retain an interruption code which provides the reason for the interruption. Since the old PSW is accessible to the program as well as to hardware, the program can examine this interruption code to determine what action is required.

A privileged instruction called Load Program Status Word (LPSW) provides the EXC program with the capability to load a PSW into the current PSW hardware from anywhere in storage. Thus, after handling an interruption, the EXC program can return to the original instruction sequence simply by loading a PSW from the old PSW where status was originally saved. For example, in the supervisor call interruption, the EXC program would do an LPSW from the location of the supervisor call old PSW to return to the original instruction stream.



The contents of the PSW are described later in this chapter. For the present, it is pointed out that each PSW contains program state bits, one of which indicates whether the CE is in supervisor state or not. Only in supervisor state can privileged instructions be executed. Since LPSW is a privileged instruction, the EXC program can load PSWs for subprograms so that they do not run in supervisor state, reserving that state for itself. It also is pointed out here that provision is made in the PSW for masking interruptions. That is, a mask field in the PSW with bits corresponding to interruption types may be used either to permit an interruption to occur or to prevent it from occurring (mask it off). This allows interruptions to be handled in an orderly fashion and avoids the condition in which one interruption follows immediately after another before the first can be handled.

The five classes of interruptions provide the EXC program with a high degree of control over system operation and with the capability to monitor for error and abnormal conditions. Each interruption class is discussed here in terms of the special facilities it provides to the EXC program. Masking is not considered here. A complete introduction to interruptions is included later in this section.

Machine-Check Interruptions

A machine-check interruption is caused by machine-checking circuits detecting a machine malfunction. A CE automatically logs the contents of various triggers and registers into a logout area in the PSA when a machine check occurs. At completion of the logout, a machine-check interruption takes place so that the control program can analyze the logout and take appropriate action.

A machine check condition in an IOCE also results in a logout into the PSA of the controlling CE, but the IOCE must first request permission to log out and to interrupt. Facilities are also available to enable a CE to request logout information from SEs and DEs when the need arises.

The machine-check interruption facility alerts the control program to malfunctions in its immediate subsystem; the logout facility provides additional information about the malfunction so that the machine-check handling routine in the control program can initiate appropriate action. For example, the control program may elect either to replace the failing element through system reconfiguration or simply to record the error and restart the program operation.

Program Interruptions

A program interruption occurs when certain conditions (such as incorrect operands or programming errors) are encountered in a program. The exact error is indicated in the interruption code of the program old PSW. Program interruptions alert the control program to program errors within itself or within subprograms for which it is responsible. Depending on the cause of the interruption, the instruction being processed may be suppressed, terminated, or allowed to continue.

Supervisor Call Interruptions

The supervisor call interruption results from the execution of the Supervisor Call (SVC) instruction. The interruption code is used to convey the reason for the SVC. In this way, a subprogram may call upon the control program to perform various services such as initiation of I/O instructions, handling of unusual conditions encountered by the subprogram, allocation of additional storage, etc.

External Interruptions

The external interruption facility of the 9020 systems is quite elaborate because it is the primary means available to the EXC program of monitoring for malfunctions and abnormal conditions in the entire system. External interruptions result from four sources, which are described in the following paragraphs.

Interval Timer at Limit. The interval timer is a fullword at location 50 hex in the PSA, which is automatically fetched from storage approximately every 16.7 ms, decremented, and stored back again. It can be used for keeping the correct time or for timing intervals between events. When the timer overflows from a positive to a negative value, an external interruption occurs. This permits the program to reinitialize the timer values.

Interrupt Pushbuttons. An external interruption may be manually initiated from a pushbutton on the CE control panel or from a pushbutton on the system or configuration console.

Direct Control. CEs and IOCE-processors can initiate external interruptions via direct control. These are gated by the CCR in the CE so that they will not be accepted if the CE is not configured to the interrupting element.

A special register, the processor interrupt register (PIR), is used in the CE to identify an interrupting IOCE-processor. A PIR bit in the external old PSW interruption code alerts the program to examine the PIR. CEs sending direct control interruptions are identified directly in the interruption code.

Abnormal Condition Signals. For external interruptions resulting from abnormal condition signals, the interruption code is not sufficient to code all the possibilities. Its function is expanded by a register called the diagnose accessible register (DAR) in the same way as it is expanded by the PIR for IOCE-processor interruptions. As stated previously, facilities are available in the PSW for masking off interruptions. These facilities, too, are expanded by a register, the DAR mask register (DARM). This is discussed later in this section under "Interruption Masking".

Bits are set in the DAR of a CE by abnormal condition signals received from CEs (including itself), SEs, IOCEs, TCUs, and PAMs or RCUs. The abnormal condition signals include: 'element check' (ELC), 'out-of-tolerance check' (OTC), and 'on battery' signal (OBS). In general, an ELC indicates a malfunction in the logic of an element, an OTC indicates an overtemperature condition, and OBS indicates that an element has switched over to battery backup power. For details regarding the meaning of these signals, as sent by different elements, refer to Chapter 8 of the system introduction manuals cited at the beginning of this chapter. The last point, OBS, should be clarified here, however.

Each major element is equipped with a battery backup power source capable of sustaining operation for approximately 5 seconds (in the event of loss of external power). Receipt of the OBS signal permits the EXC program to close out operation of an element (or the whole system if all power is lost), in a convenient and orderly manner, in preparation for a power-down condition at the end of the 5 seconds. In this way, checkpoints can be established by the program so that very little reinitialization is required when power is restored.

Receipt of an abnormal condition signal from a given element sets a particular bit in the DAR and, provided all masking conditions are met, initiates an external interruption. A DAR bit in the external old PSW alerts the control program to examine the DAR and determine the exact cause of the interrupt.

Input/Output Interruptions

I/O interruptions provide a means by which IOCEs can notify their controlling CE of special conditions concerning the attached I/O devices. The address and channel of the interrupting device is made available in the I/O old PSW interruption code.

Responsibilities of the Control Program

We have examined the special facilities that are available to the control program and which enable it to monitor and exert control over total system operation. Now, the various responsibilities of the control program may be summarized. The major responsibilities of the control program are listed as follows:

1. Configuring the system into subsystems appropriate for the current tasks and, also, reconfiguring as conditions vary.
2. Assigning SEs to subsystems and initializing the address translation hardware in the CEs and IOCEs.
3. Establishing PSAs and alternate PSAs and initializing PSBAR in the CEs.
4. Loading and operating subprograms.

5. Sharing areas of main storage among subprograms needing access to common information.
6. Scheduling and controlling I/O operations.
7. Providing services to subprograms, when requested via supervisor calls.
8. Coordinating multisystem operation.
9. Monitoring for errors, malfunctions, and other abnormal or exceptional conditions.
10. Keeping a running log of machine checks and I/O errors.
11. Providing standard error- and malfunction-handling routines, such as error analysis and reconfiguration.
12. Accommodating manually entered requests for maintenance elements or subsystems when conditions permit.
13. Interfacing with operating personnel via operational messages.

ELEMENT STATES

Each major element and unit in the 9020 systems can operate in any one of four element states, depending on the state bit settings in the CCR. The state of an element largely determines its availability to the ATC system because it affects the element's response to reconfiguration and the degree to which manual control over the element is permitted. In the case of the CE, the capability to execute the SCON instruction effectively and the capability to mask ELCs received from other CEs is also affected by the state.

There are four element states, 0 through 3. State 3 is the highest operational state; i.e., it is the state in which the EXC program has maximum control of the element and manual control is minimal. State 0 is the lowest operational state, with maximum manual control permitted.

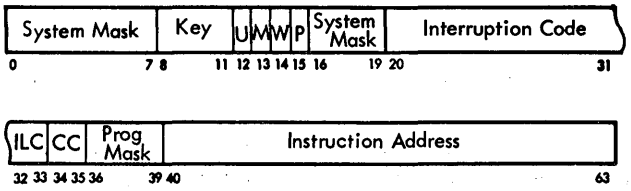
State 0 is divided into two substates by a TEST switch on each element (including the CE). The major effects of placing an element in 'test' (turning the TEST switch on) are preventing the element from being "configured away" by some CE executing a SCON instruction and enabling maintenance personnel to power the element up and down.

Table 1-1 summarizes the effects of element state on a CE. Table 4-1 in Chapter 4 shows how manual controls are affected by state and by the setting of the TEST switch. In connection with the enabling and disabling of manual controls, one other condition must be considered. That is the setting of the SYSTEM INTERLOCK switch. This locking switch enables certain CE panel controls to permit SC or CC functions to be duplicated at the CE.

A similar SYSTEM INTERLOCK switch at the system or configuration console permits a number of CE panel control functions to be initiated from the console. The SYSTEM INTERLOCK switch at the console overrides the switch on the CE panel if both are turned on.

PROGRAM STATUS WORD

A doubleword, the program status word (PSW), contains the information required for proper program execution. In general, the PSW controls instruction sequencing and holds and indicates the status of the CE in relation to the program being executed. The PSW has the following format:



Bits 0–7 and 16–19, System Masks. Associated with I/O and external interruptions as follows:

System Mask Bit	Interruption Source
0	Multiplexer channel 0
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Multiplexer channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, Interrupt pushbuttons, Direct Control, PIR, and DAR
16	Selector channel 7
17	Multiplexer channel 8
18	Selector channel 9
19	Selector channel A

Note: Channels 0–3 = IOCE 1; channels 4–7 = IOCE 2; channels 8–A = IOCE 3

When a system mask bit is 1, the associated source can interrupt the CE; when bit is a 0, the interruption remains pending.

Bits 8–11, Key. Contain the CE storage protection key. The key is matched with the storage key whenever data is stored or whenever data is fetched from a location that is fetch-protected.

Bit 12, U (USASCII-8). Affects decimal operations only. When bit is a 1 and in unpacked format, decimal digits are represented by USASCII-8. When bit is a 0, EBCDIC is specified.

Bit 13, M (Machine-Check Mask). When bit is a 1, a machine check causes the CE to log out its status to main storage and to take a machine-check interruption. If the machine-check mask is a 0, the machine check remains pending. As a maintenance aid, the CHECK CONTROL

Table 1-1. Effect of Element State on CE Operation

CE State	Execute SCON Instruction	Issue SCON (Note 3)	Accept SCON (Note 4)	CE ELC (Note 6) Maskable	CE Panel Controls (Note 7)			System or Configuration Console Controls
					Operator	Maintenance	CCR, Power	
Three	Yes	Yes	Yes	Yes	No	No	No	Yes (Note 8)
Two	No (Note 1)	N/A	Yes	No	No	No	No	Yes
One	No (note 1)	N/A	Yes	No	Yes	No	No	Yes
Zero, and Test Switch Off	Yes	Yes	Yes	No	Yes	Yes	No	Yes
Zero, and Test Switch On	Yes (Note 2)	No (Note 3)	No (Note 5)	Yes	Yes	Yes	Yes	No

Notes:

(1) Execution of Set Configuration instruction is suppressed and a specification interruption is taken.

(2) Manual control of CCR settings is provided. The Set Configuration instruction may be executed when the state bits are manually set to Three or Zero. The selection of other system elements and the propagation of the configuration mask are suppressed. All instruction exceptions are recognized and program interruptions are taken. If Set Configuration instruction is attempted while the state bits are set to One or Two, note (1) applies.

(3) "Issue SCON" refers to the ability to select external elements and present the configuration mask information when executing the Set Configuration instruction.

(4) "Accept SCON" means to accept the configuration mask bits into the CCR. The Scon bit for the issuing CE must be already set in the receiving CCR. If either the Scon bit is off or the mask is accepted with improper parity, condition code 2 is set in the issuing CE. (Two important exceptions to this rule should be noted: Whenever any receiving element has detected improper parity on the content of its CCR, or has its state bits set to Three, Two, or One and has all scon bits set off in its CCR, configuration mask bits are unconditionally accepted from any issuing CE.)

(5) All external configuration selection and mask signals are ignored. A self-issued configuration mask is accepted provided its own scon bit is already set in the CCR.

(6) Incoming CE ELC's (element checks) are masked off when the Scon bit for the issuing CE is off in the CCR of the receiving CE. Where "yes" is entered the CE ELC is also maskable by the external interruption mask, PSW bit 7. "No" indicates that any additional masking provided is ignored. The receiving CE has its state bits set to Three, and an abnormal condition interruption is taken. Refer to chapter 7 of 9020 System Introduction Manuals for details.

(7) A limited number of operator's controls on the CE control panel are activated. Refer to Table 4-1 in Chapter 4.

(8) A number of the CE control panel facilities are enabled on the system console and configuration console.

switch on the system control panel can modify the machine-check mask bit functions when the CE is in state 0. This bit also masks the IOCE special machine-check interruption.

Bit 14, W (Wait State). When bit is a 1, CE is in the Wait state; instructions are not executed until an external or I/O interruption or an IPL occurs. When bit is a 0, the CE is in the Running state.

Bit 15, P (Problem State). When bit is a 1, CE is in the Problem state. When bit is a 0, CE is in the Supervisor state.

Bits 20–31, Interruption Code. Identify the cause, purpose, or source of the interruption. (Code has no meaning in a new PSW.)

Bits 32 and 33, ILC (Instruction Length Code). Indicate the length, in halfwords, of the last processed instruction. This code is predictable only for most program and supervisor call interruptions. For I/O and external interruptions, the interruption is not caused by the last interpreted instruction, and the code is not predictable for these instructions. For machine-check interruptions, the setting of the code may be affected by the malfunction and, therefore, is unpredictable. A code of 0, used only for program interruptions, indicates that the instruction address in the PSW is not the location of the instruction following the instruction that caused the program interruption.

Bits 34 and 35, CC (Condition Code). Contain the condition code that reflects the result of most arithmetic, logical, or I/O instructions. Each of these operations can set the code to any one of four states, and the conditional-branch instructions can specify the state to be used as the criterion for branching. For example, the CC may reflect

such conditions as nonzero, overflow, and underflow. Once set, the CC remains unchanged until modified by an instruction that reflects a different code. The two bits of the CC provide for four possible binary settings: 00, 01, 10, and 11. This manual refers to the CCs as 0, 1, 2, and 3. (The CC has no meaning in a new PSW.)

Bits 36–39, Program Mask. Each bit is associated with a program interruption as follows:

Program Mask Bit	Program Interruption
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

When a program mask bit is a 1, the associated program interruption results in an interruption; when bit is 0, the interruption is lost.

Bits 40–63, Instruction Address. Specify the leftmost byte address of the next instruction.

The active or controlling PSW is called the current PSW. The information making up the current PSW is held in triggers and registers in the CE and is constantly updated as instructions are executed. (The instruction-address field of the PSW, for example, is held in the instruction counter and is updated to give the address of the next instruction to be executed.) When an interruption is taken, the PSW is assembled in the ST register and is transferred to a fixed location in main storage corresponding to the interruption. (The stored PSW is called the old PSW.) In this way, the program can preserve, for subsequent analysis, the status of the CE at the time of interruption. To complete the interruption, a new PSW is introduced into the CE from another unique main storage location, and the instruction at the specified location is fetched.

In certain circumstances, the entire PSW is loaded into the CE; in others, only part of it. As explained in the preceding paragraph, when the CE is interrupted, the entire current PSW is stored, and an entire new PSW is loaded. The state of the CE may be changed by executing the Load PSW instruction, which introduces a new PSW. New program mask and system mask bits may be specified by altering the corresponding fields in the PSW through the Set Program Mask and Set System Mask instructions, respectively. The Set Program Mask instruction also changes the condition code.

Program States

There are eight paired states, Problem/Supervisor, Operating/Stopped, Running/Wait, and Interruptable/Masked. All of these states, except Stopped, are defined in the PSW. The following paragraphs define the program states and discuss the PSW. The eight program states that determine the overall CE status differ in he way they affect the CE functions and in the way their status is indicated. Refer to Table 1-2 for pertinent information about the program states.

Table 1-2. Program States

State	Comments
Problem	Load PSW, Set System Mask, Diagnose, Load PSBAR, Store PSBAR, Set Configuration, Set Address Translator, and Start I/O Processor instructions, and all I/O, storage protection, and direct control instructions are invalid. These instructions are termed privileged instructions.
Supervisor	All instructions are valid.
Operating	CE processes instructions (if not in Wait state) and interruptions (if not masked off). Entered by: 1. Depressing START at SC or CC with CE selected and SYSTEM INTERLOCK on. 2. Depressing START on CE control panel with CE in state zero or with SYSTEM INTERLOCK on at CE and off at SC or CC. 3. Starting an IPL operation.
Stopped	Instructions and interruptions are not processed; interruptions remain pending. The timer is not updated. Execution of program is not affected by stopping CE. Entered by: 1. Depressing STOP at SC or CC with CE selected and SYSTEM INTERLOCK switch on or, depressing ALL STOP at SC or CC with SYSTEM INTERLOCK switch on. 2. Depressing STOP on CE control panel with CE in state zero or with SYSTEM INTERLOCK on at CE and off at SC or CC. 3. Detecting equality on an address-compare stop operation with CE in state zero or one with SYSTEM INTERLOCK on at CE and off at SC or CC. 4. Completing one instruction when in instruction-step mode with CE in state zero or one with SYSTEM INTERLOCK on at CE and off at SC or CC. 5. Turning power on or following a system reset with CE in state zero and TEST switch on.
Running	Instruction processing proceeds in normal manner.
Wait	No instructions are processed, and main storage is not addressed. The CE waits for an I/O or external interruption to occur before executing further instructions, or for an IPL operation.
Interruptable	Interruptions are accepted.
Masked	System and machine-check interruptions remain pending, and program interruptions are ignored.

Problem/Supervisor

In the Problem state, all privileged instructions are invalid. A privileged instruction encountered in the Problem state constitutes a privileged-operation interruption and interrupts the operation. In the Supervisor state, all instructions are valid. Because of the Problem/Supervisor program states, programs run in Problem state (such as subprograms running under the EXC program) are often called problem programs and programs like the EXC are often called supervisor programs.

The CE is switched between the Problem and Supervisor states by changing PSW (15). When PSW (15) is a 1, the CE is in the Problem state; when a 0, the CE is in the Supervisor state. This bit can be changed only by introducing a new PSW. Thus, the status switching for Problem/Supervisor state may be performed by an interruption operation or by a Load PSW instruction containing a new PSW with the desired value in bit 15. Because the Load PSW instruction is a privileged instruction, the CE must be in the Supervisor state before the switch. The CE status can also be changed between Problem and Supervisor states by issuing a Supervisor Call instruction or an initial program load (IPL). The Supervisor Call instruction causes an interruption which will load new PSW data. This new PSW data may change the state of the CE. Similarly, the IPL introduces a new PSW. The new PSW may introduce the Problem or Supervisor state, regardless of the preceding CE state.

Operating/Stopped

When the CE is in the Stopped state, instructions and interruptions are not executed. When the CE is in the Operating state, instructions are executed as long as the CE is not also in the Wait state. Interruptions are taken if they are not masked off. A change in the Stopped/Operating states can occur only by manual intervention or by machine malfunction. No instruction or interruption can start or stop the CE. The CE is placed in the Stopped state when STOP on the CE control panel is depressed, detecting equality on an address-compare-stop operation, completing one instruction when in instruction-step mode, and after power is turned on or following a system reset, except during IPL. For these manual controls to be enabled, the CE must be in element state 0, 1, or test; or must have the SYSTEM INTERLOCK switch (at the CE control panel) turned on while the SYSTEM INTERLOCK at the SC or CC is turned off. The CE may also be placed in the Stopped state from the SC or CC by turning on the SYSTEM INTERLOCK switch there and depressing ALL STOP or STOP with that CE selected.

The CE is placed in the Operating state by depressing START or LOAD on the CE control panel, provided the CE is either in state 0, 1, or test; or if the SYSTEM INTERLOCK switch is turned on at the CE control panel and off at the SC or CC. The CE is also placed in the operating state if START or LOAD is depressed at the SC or CC with the CE selected and the SYSTEM INTERLOCK switch on at the SC or CC.

Changing from the Operating state to the Stopped state occurs at the end of instruction execution and before the start of the next instruction execution. When the CE is in the Wait state, the change from Operating to Stopped occurs immediately. All interruptions pending and not masked off are taken while the CE is still in the Operating state. The interruptions cause an old PSW to be stored and a new PSW to be fetched before entering the Stopped state; interruptions are no longer taken but remain pending.

Running/Wait

In the Wait state, no instructions are processed, and main storage is not addressed. In the Running state, instruction fetching and execution proceed in the normal manner. The CE status is switched between the Wait and Running states by PSW(14). When PSW(14) is a 1, The CE is in the Wait state; when a 0, CE is in the Running state. This bit can be changed only by introducing a new PSW. Thus, switching from the Running to the Wait state may be achieved by the privileged instruction Load PSW, by an interruption such as given by a Supervisor Call instruction, or by an IPL. Switching from the Wait to the Running state may be achieved by an I/O or external interruption or by an IPL. The new PSW may introduce the Wait or Running state regardless of the preceding CE state.

Interruptable/Masked

The Interruptable/Masked state of the CE is determined by the system mask bits PSW(0–7, 16–19), the machine-check mask bit PSW(13), and the program mask bits PSW(36–39). If a mask bit is a 1, the associated interruption is accepted; if it is a 0, system and machine-check interruptions remain pending, and program interruptions are ignored. The PSW bits and interruptions that will occur if the bit is active are listed in Table 1-3.

The Interruptable/Masked state of the CE is switched by changing the mask bits in the PSW. The program mask may be changed separately by the Set Program Mask instruction, and the system mask may be changed separately by the Set System Mask instruction. The machine-check mask bit can be changed only by introducing an entirely new PSW, as in the Problem/Supervisor and Wait/Running states. Thus, a change in the entire masked status may be achieved by the privileged instruction Load PSW, by an interruption (such as for the Supervisor Call instruction), by an IPL, or by a

Table 1-3. PSW Interruption Mask Bit Designations

PSW Bit	Interruptions
System Mask	
0	Multiplexer Channel 0
1	Selector Channel 1
2	Selector Channel 2
3	Selector Channel 3
4	Multiplexer Channel 4
5	Selector Channel 5
6	Selector Channel 6
7	Timer, INTERRUPT Pushbuttons, External Signals, DAR, or PIR
16	Selector Channel 7
17	Multiplexer Channel 8
18	Selector Channel 9
19	Selector Channel A
Machine-Check Mask	
13	Machine-Check
Program Mask	
36	Fixed-Point Overflow
37	Decimal Overflow
38	Exponent Underflow (Floating-Point)
39	Significance (Floating Point)

PSW restart operation. Regardless of the preceding program state, the new PSW may introduce a new mask status.

Interruption Masking

Sometimes it is not desirable to allow an interruption. This condition becomes apparent when I/O interruptions are considered (Figure 1-2). Assume an I/O operation is completed, resulting in an I/O interruption. The current PSW is stored as the old PSW to give the supervisor the reason for (or the source of) the interruption. This old PSW also enables the supervisor to return to the interrupted problem program. A new PSW is then brought out of storage to become the current PSW, which indicates the first instruction of the I/O interruption-handling routine. At this time, if a second I/O interruption (perhaps caused by operator intervention at an I/O device of another channel) were allowed, the old PSW stored as a result of the first I/O interruption would be lost. The supervisor can prevent this second I/O interruption from being accepted until it has processed the first I/O interruption by means of mask bits in the new PSW. If the corresponding mask bit is a 1, the interruption is taken; if bit is a 0, the interruption is ignored or remains pending. External and I/O interruptions may be masked off by the system mask field of the PSW; machine-check interruptions may be masked off by the machine-check mask bit; 4 of the 15 program interruptions may be masked off by the program mask field.

System Mask Field. The system mask field consists of 12 bits PSW(0–7, 16–19), which can be used selectively or collectively to mask all I/O and external interruptions. These are shown in Table 1-3.

To prevent an I/O or external interruption before the first interruption has been processed, the system mask of the new PSW should contain 0's. When a system mask bit is a 0, the associated I/O or external interruption remains pending.

The system mask field may be changed by introducing a new PSW, or it may be changed separately by the Set System Mask instruction.

External interrupts resulting from bits set in DAR by abnormal condition signals are masked by corresponding bits in the DAR mask register. These bits are set via a privileged instruction called Diagnose. Refer to Chapter 7 of the system introduction manuals cited at the beginning of this chapter for further discussion.

Machine-Check Mask Bit. The machine-check mask bit PSW(13) controls the acceptance of a machine-check interruption. If this bit is a 0, machine-check interruptions are ignored and remain pending. If this bit is a 1, machine-check interruptions are taken, depending on the position of the CHECK CONTROL switch on the CE control panel. If this switch is in the PROC (normal) position, the CE stops, and the status is logged into main storage; a machine-check interruption then takes place. If the CE CHECK CONTROL switch is in the DSBL (disable) position, the CE does not stop upon detection of a machine check, and no logout or interruption takes place. If the switch is in the STOP position, the CE stops upon detection of a machine check, but there is no logout of data, and no interruption takes place.

The usual mode of operation is to have the CHECK CONTROL switch set to the PROC position and PSW(13) set to a 1.

The machine check mask bit can be changed only by introducing a new PSW.

Program Mask Field. The program mask field consists of four bits, PSW(36–39), each of which is associated with a program check:

Program Mask Bit	Program Interruption
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

When a program mask bit is a 1, the associated program check results in an interruption; when a 0, no interruption occurs, and the condition does not remain pending.

The program mask field may be changed by introducing a new PSW, or it may be changed separately by the Set Program Mask instruction.

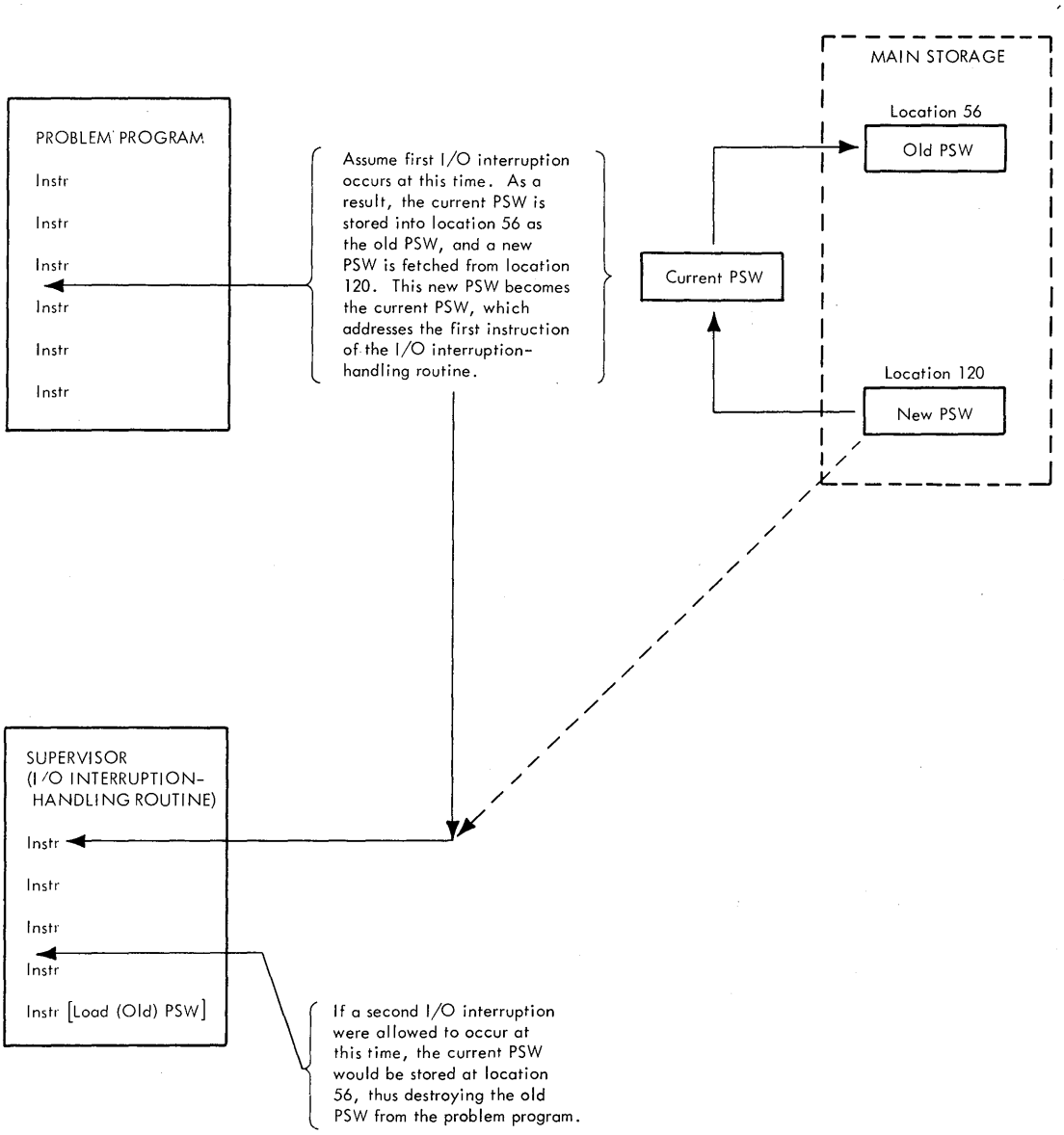


Figure 1-2. Example of Need for Interruption Masking

PREFERENTIAL STORAGE AREA

Each CE in a 9020 system has a 4096-byte area of its assigned storage set aside for the storage of PSWs, channel control words (CAW and CSW), interval timer, and the diagnostic logout area. This area is called the preferential storage area (PSA). Table 1-4 shows those contents of the PSA that must be at fixed locations within the PSA so that they are accessible to hardware. The remaining portion of the 4096-byte area may be used at the discretion of the programmer.

Programs written for the 9020 are based on the assumption that the PSA is in the lowest 4096 bytes of storage. Thus, referring to Table 1-4, the CSW is accessed by addressing the doubleword at the absolute address of hex 40. This assumption is valid provided the control program has properly initialized the PSBAR in the CE that is executing the program. Hardware in the CE monitors for addresses having the 12-high-order bits 0's. Such an address indicates that the program is attempting to access the PSA. When such an address is detected, high-order bits from PSBAR are ORed with the address, and the location in the actual PSA is accessed.

The IOCE is often required to access its controlling CE's PSA. For example, the CE's PSA must be accessible to the IOCE for I/O operations, interruptions, and an IOCE logout. A PSBAR in the IOCE is updated by the CE, via a bus called the control bus, whenever the CE requests the IOCE to perform an I/O operation (including IPL) or grants permission to interrupt on logout.

Table 1-4. Preferential Storage Areas

Main Storage Address		Length	Information Stored
Dec	Hex		
0	0	Doubleword	Initial program loading PSW
8	8	Doubleword	Initial program loading Channel Command Word 1 (CCW 1)
16	10	Doubleword	Initial program loading CCW 2
24	18	Doubleword	External interruption, old PSW
32	20	Doubleword	Supervisor call interruption, old PSW
40	28	Doubleword	Program interruption, old PSW
48	30	Doubleword	Machine check interruption, old PSW
56	38	Doubleword	I/O interruption, old PSW
64	40	Doubleword	Channel Status Word (CSW)
72	48	Word	Channel Address Word (CAW)
76	4C	Word	Unassigned
80	50	Word	Timer
84	54	Word	Unassigned
88	58	Doubleword	External interruption, new PSW
96	60	Doubleword	Supervisor call interruption, new PSW
104	68	Doubleword	Program interruption, new PSW
112	70	Doubleword	Machine check interruption, new PSW
120	78	Doubleword	I/O interruption, new PSW
128	80	64 doublewords	Diagnostic log-out area

Note that an IOCE-processor has a PSA also, but it is located in an internal storage, called MACH, which is not part of main storage and is unrelated to PSBAR.

CONTROL OF I/O OPERATIONS

The following paragraphs discuss how I/O operations are controlled by instructions, commands, orders, and control words; and illustrate how the I/O system works, using the Start I/O instruction as an example.

Instructions, Commands, and Orders

Input/output operations are initiated and controlled by three types of information: instructions, commands, and orders. Instructions are decoded by the CE and are part of the CE program. Commands are decoded and executed by the IOCE, and they initiate I/O operations such as reading and writing. Instructions and commands are fetched from main storage and are common to all types of devices. Orders specify functions peculiar to an I/O device, such as rewinding tape or spacing a line on a printer. Orders are contained in the control command; they are decoded and executed by the device.

The action in an I/O device initiated by a command is termed an I/O operation. Five I/O operations are available: write, read, read backward, control, and sense. The IOCE channel initiates the operation by executing the associated command.

The Write command initiates a write operation at the device. Data from main storage is fetched in an ascending order of addresses and transferred to the device.

The Read command initiates a read operation at the device. Data is read from the device in the same sequence as it was written by a Write command. Data is placed into main storage in an ascending order of addresses.

The Read-Backward command initiates a read-backward operation at the device. Data is read from the device in a sequence opposite to that in writing. Data is placed into main storage in a descending order of addresses.

The Control command contains information, termed orders, that controls the selected device. Orders are unique to the particular device in use and specify such functions as backspacing or rewinding magnetic tape. Orders are fetched from main storage in an ascending order of addresses and transferred to the device.

The Sense command initiates a sense operation at the device. Data transferred during a sense operation provides

information about unusual conditions detected during the last operation and the status of the device. Data is placed into main storage in an ascending order of addresses.

I/O Control Words

Three I/O control words are used during an I/O operation:
1. Channel address word (CAW), which initiates I/O sequencing.
2. Channel command word (CCW), which controls I/O operations and sequencing.
3. Channel status word (CSW), which indicates channel status.

Channel Address Word

The CAW specifies the address of the first CCW associated with the Start I/O instruction. The CAW is assigned permanent main storage address 72 (decimal). The IOCE channel refers to the CAW only during execution of the Start I/O instruction. The pertinent information is stored in the IOCE channel, and the CE program is free to change the contents of the CAW. The CAW has the following format:

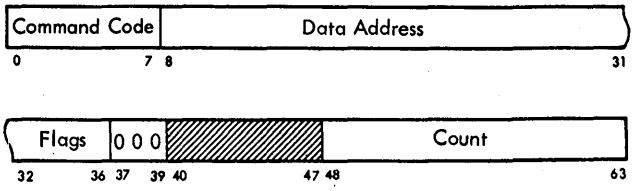
Bits 0–3 Key. Specifies the storage protection key for all commands associated with the Start I/O instruction.

Bits 4–7. Must be all 0's.

Bits 8–31, Command Address. Designates location of the first CCW in main storage.

Channel Command Word

The CCW specifies the command to be executed and, for commands initiating I/O operations, designates the main storage area associated with the operation and the action to be taken whenever data transfers to or from main storage are completed. The CCWs can be located anywhere in main storage, and more than one can be associated with a Start I/O instruction. The channel refers to a CCW in main storage only once, whereupon the pertinent information is stored in the channel. The first CCW is fetched during execution of the Start I/O instruction. Each additional CCW is obtained when the operation has progressed to the point where the additional CCW is needed. The CCW has the following format:



Bits 0–7, Command Code. Specifies I/O operation to be performed.

Bits 8–31, Data address. Specifies location of an eight-bit byte in main storage; it is the first location referred to in the main storage area designated by the CCW.

Bits 32–36, Flags. Cause certain functions to be performed that modify the operation.

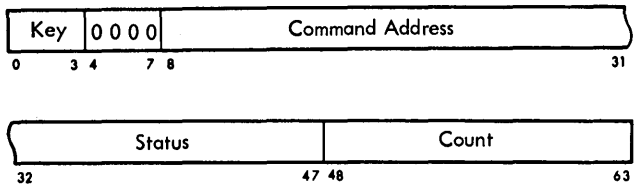
Bits 37–39. Must be all 0's for every CCW other than the CCW that specifies a transfer-in-channel operation.

Bits 40–47. Not used.

Bits 48–63, Count. Specifies the number of eight-bit byte locations in the main storage area designated by the data-address field in the CCW.

Channel Status Word

The CSW provides the program with the status of an I/O device or the condition under which an I/O operation has been finished. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and during execution of the Start I/O, Test I/O, and Halt I/O instructions. The CSW is placed into main storage location 40 hex and is available to the program at this location until the next I/O interruption occurs or until another I/O instruction causes its contents to be replaced, whichever occurs first. The CSW has the following format:



Bits 0–3, Key. Contains the storage protection key that was used in the I/O operation initiated by the last Start I/O instruction.

Bits 4–7. Must be all 0's.

Bits 8–31, Command Address. Identifies the last CCW used.

Bits 32–47, Status. Identifies the conditions in the I/O device and channel that caused the CSW to be stored.

Bits 48–63, Count. Contains the residual count of the last CCW used.

I/O System Operation

The CE program initiates I/O operations by means of the Start I/O instruction. The instruction is decoded by the CE, and the following data is placed on the CE-IOCE control bus: unit and channel address, I/O op code, and PSBAR. The IOCE is then signaled via a line called 'I/O instruction'. This instruction identifies the I/O device and causes the IOCE channel to fetch the CAW from main storage location 48 hex. The CAW designates the location in main storage from which the channel subsequently fetches the first CCW. The CCW specifies the command to be executed, the main storage area to be used, and the number of data bytes to be transferred, if any.

The channel attempts to select the device by sending the address of the device to all attached control units. Upon recognizing the address, the control unit associated with the addressed device connects itself logically to the channel. The channel subsequently sends the command code to the device, and the device responds by indicating whether it can execute the command.

At this time, the IOCE releases the CE, and execution of the Start I/O instruction is terminated. The CE then continues with its program. The results of the attempt to initiate command execution are indicated in the PSW condition code and, under certain conditions, by storing a portion of the CSW.

If the operation is initiated by the I/O device and its execution involves transfer of data, the channel responds to service requests from the device and assumes control of the operation. For operations that do not require transfer of data, the device signals the end of the operation immediately on receipt of the command code, and the channel is immediately available for a new I/O operation.

An I/O operation may involve transfer of data to or from one main-storage area, designated by a single CCW, or, when data chaining is specified, to or from a number of noncontiguous main-storage areas. In the latter case, a chain of CCWs is used in which each CCW designates an area in main storage for the continuation of the original command (operation).

Termination of the I/O operation normally is indicated by two conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all information associated with the operation and no longer needs channel facilities. The device-end condition indicates that the device has finished the operation.

Facilities are provided for the program to initiate execution of a chain of commands with a single Start I/O instruction. When command-chaining is specified, the device-end condition causes the channel to fetch a new CCW that specifies a new operation at the device.

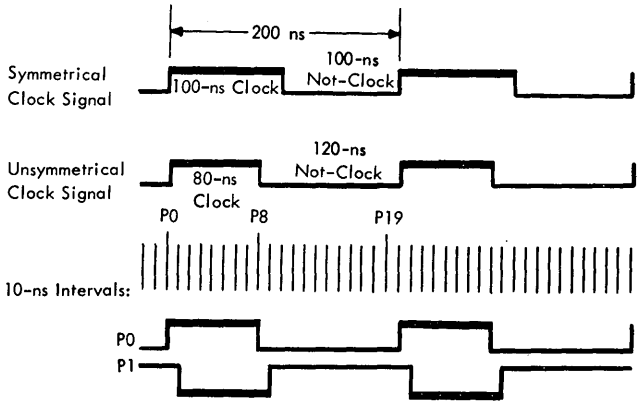
Conditions that initiate I/O interruptions are asynchronous with the activity in the CE, and more than one interruption condition can occur at the same time. A priority has been established among the conditions so that only one interruption is processed at a time. The I/O interruption conditions are preserved in the I/O devices and channel until accepted by the CE.

This section discusses (1) CE control and data transfer, (2) instruction fetching and execution, (3) the instruction set by instruction class, (4) interfacing and intercommunication between the CE and other elements in the system, and (5) maintenance features and power considerations.

TIMING

- Basic clock cycle period is 200 ns.
- Symmetrical clock signal consists of 100-ns clock portion and 100-ns not-clock portion.
- Unsymmetrical clock signal consists of 80-ns clock portion and 120-ns not-clock portion.

The basic CE clock cycle period is 200 ns, divided into clock and not-clock portions. A clock signal generator provides a 5-mHz symmetrical (100-ns/100-ns) clock signal. To provide additional time for CE logic functions, the symmetrical clock signal is modified to give a 5-mHz unsymmetrical (80-ns/120-ns) clock signal. Finer intracycle control is obtained by dividing each of the two clock signals into 20 intervals of approximately 10 ns each. These intervals, named P0, P1, P2,...P19, are created by inverters which delay the signal by about 10 ns. Thus, the notations P0, P1, P2,...P19 refer to signals which are inverted and are delayed 10 ns with respect to the previous signal:



The clock signals are distributed to logic gates A through L. Adjustable time delays within the logic gates synchronize the clock signals with a reference signal, thereby eliminating the various amounts of delay introduced by the distribution cables. The distribution of the clock signals to the CE processing logic is stopped upon detection of an error or during scan, logout, single-cycle, and certain ROS operations. During maintenance operations (such as scan, logout, and single-cycle operations) the clock signals may be stopped or permitted to run intermittently.

DATA TRANSFER

Data is transferred into a register, into an adder, and into and out of LS by gating signals controlled by ROS (Figure 1-16). Referring to Figure 1-3, note that data from PAL is always available at the A-register, B-register, and T-register, but is transferred only into the selected register by means of the corresponding gating signal from ROS. When gating data into an adder, timing considerations require the use of 'gate control' triggers; these triggers, which are set by the ROS decode logic, generate the required gating signal.

When transferring data into LS, the gating signal from ROS is combined with a signal from the LS addressing logic to develop a 'write LS' signal, which gates the data into LS. When transferring data from the LS, an address signal selects 1 of 24 LS registers, the contents of which are transferred to the LS bus. A second gating signal transfers the data from the LS bus to the S- or T-register.

READ-ONLY STORAGE

The CE is controlled by ROS, a permanently recorded microprogram, supplemented by conventional control logic. A read-only storage is a storage device which contains information (1's and 0's) of a nondestructive nature. The 7201-02 CE utilizes a capacitive read-only storage, in which bits are stored in the form of a capacitance between a fixed drive-plate pattern etched at right angles to a sense-plate pattern. Sense and drive plates are separated by a Mylar† film (approximately 1 mil thick), and the resulting sandwich is held together under pressure. A 1-signal is coupled from a drive line to one of a pair of sense lines, and a 0-signal is coupled to the other. Sense line outputs are detected in a differential amplifier which in turn feeds a latch. When decoded, the information in ROS controls gates to route data in the CE. Access time is approximately 95 ns.

†Trademark of E. I. duPont de Nemours & Co. (Inc.)

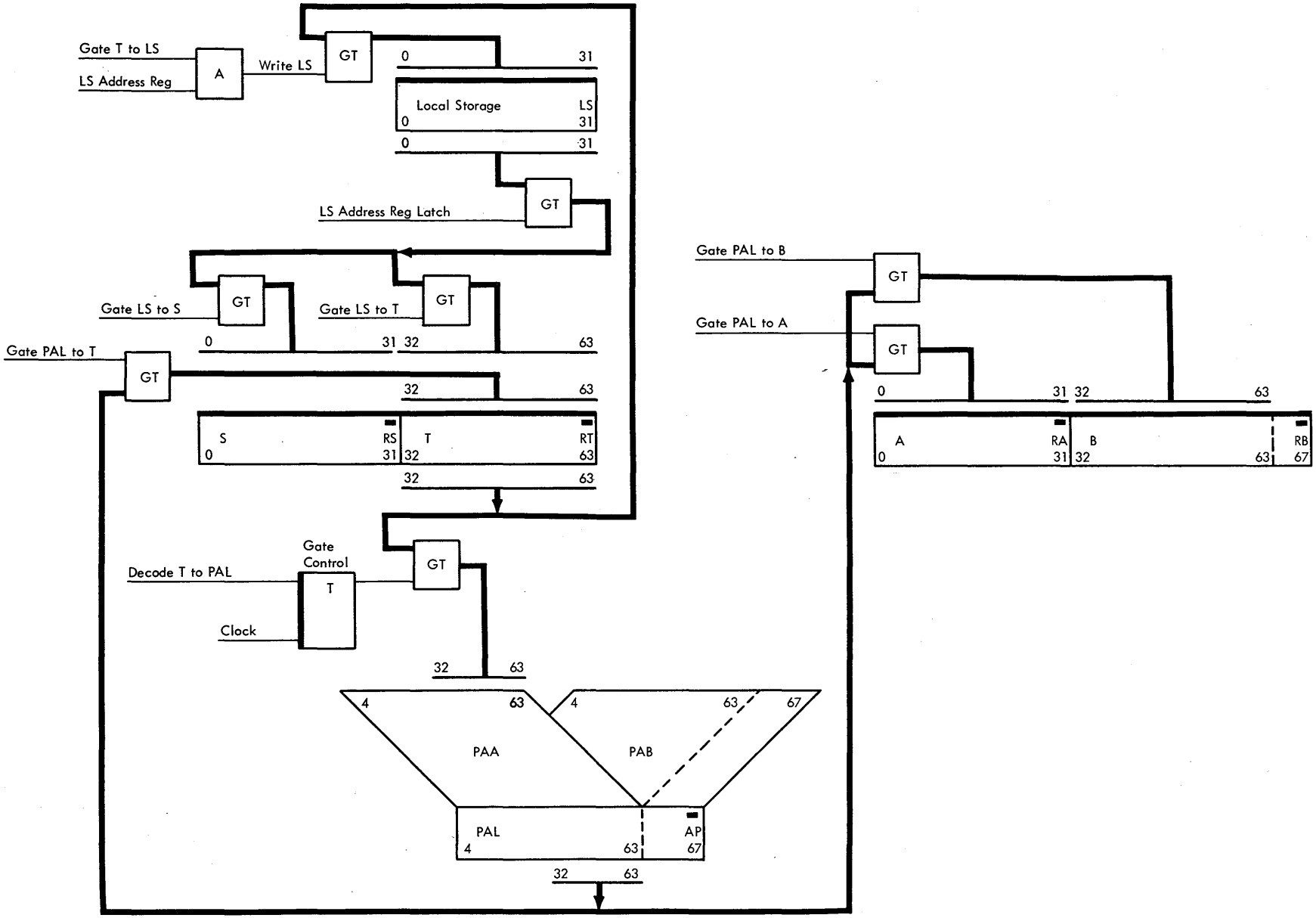


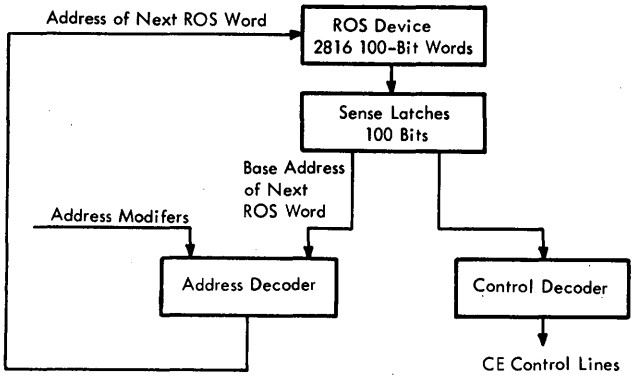
Figure 1-3. Data Transfer Scheme

Relationship of ROS to Conventional Controls

- ROS words replace most conventional sequence triggers and control lines.
- ROS word is a unique bit configuration and controls CE during machine cycle it is in use.
- In addition to control data, ROS word holds address of next ROS word and branch tests, if any.
- For branches, 1 ROS word is associated with each possible condition.

To understand ROS operation, it is helpful to note its relationship to conventional controls. Conventional controls may be characterized by sequence triggers, and by the control lines activated by the sequence triggers as a function of the operation to be performed and data conditions. Each cycle that the CE may take represents a state of the CE as defined by the control circuits. Each state, in turn, specifies which control lines are to be activated during that cycle and which state is to follow next. The defined state will cause the next sequence trigger to be set in the following cycle. In some cases, the next state may be contingent upon a branch condition in which one or two or more sequence triggers must be selected.

In a ROS-controlled processor, the sequence triggers are replaced by micro-instructions or ROS words. Each ROS word consists of a predetermined bit pattern and represents a state of the CE. A micro-instruction is read into the sense latches from the ROS device as follows:



Decoding of the bit pattern activates control lines which initiate operations or functions in the CE under timing control of ROS decode logic. The base address of the next ROS word to be used is also included in each ROS word. Data conditions within the CE may modify the address if the bit pattern indicates a test for branching (e.g., branch if overflow occurs). One ROS word is associated with each possible condition; the base address is modified to address the ROS word which satisfies the data conditions. Thus ROS eliminates the need for most of the complex sequencing networks.

ROS Word

- ROS word is divided into 100 bits, grouped into 22 control fields.
- Number of bits within field determines number of unique control signals within field.
- Control signals are termed micro-orders.

The ROS is physically organized into 16 planes, each plane holding 88 200-bit words. Through addressing, the 200-bit word is further divided in half to yield 2816 100-bit words, hereafter referred to as ROS words. Each ROS word consists of a unique predetermined bit configuration grouped into 22 control fields (Table 1-5). The number of bits within a field determines the number of unique control signals (micro-orders) available within that field. (In a four-bit field, for example, 16 distinct micro-orders can be defined, only one of which can be activated at any one time.) The micro-orders are grouped functionally within the fields according to two rules:

1. Micro-orders that are functionally similar (such as micro-orders that control ingating to the AB register) are grouped in one field for ease of decoding.
2. All micro-orders grouped in a field must be mutually exclusive because only one micro-order within that field may be specified at a time.

Usually, rule 1 results in rule 2.

When decoded, each micro-order activates one or more control lines that condition gates to perform the function specified by the micro-order. Each micro-order is assigned a mnemonic code (up to 12 characters) that defines the control function performed. For example, the micro-orders, their bit configurations and functions, associated with control field V of the ROS word are shown below.

Bit Configuration			Micro-Order Mnemonic	Function
97	98	99		
0	0	0	0	Zero gated with parity
0	0	1	E3	E(12–15) to PB(60–63)
0	1	0	E2	E(8–11) to PB(60–63)
0	1	1	E23	E(8–15) to PB(56–63)
1	0	0	Q7	Q(52–63) to PB(52–63)
1	0	1	Q5	Q(36–47) to PB(52–63)
1	1	0	Q3	Q(20–31) to PB(52–63)
1	1	1	Q1	Q(4–15) to PB(52–63)

Table 1-5. ROS Word Breakdown

Bits	Control Field	Function Controlled
0	–	Parity
1	–	Spare
2--5	W	FAA and miscellaneous control lines
6--9	A	A-, B-, and IC-register ingating
10, 11	B	LS to S- and T-register ingating
12--16	C	PSW and S-, T-, D-, G-, and Q-register ingating
17--19	D†	F-register ingating, and end-op signals
20	–	Parity
21--24	E*	E- and R-register ingating if bit 81 off; Emit field if bit 81 on.
25--30	F†	Status triggers and miscellaneous control lines
31--35	G†	Status triggers, IC, and miscellaneous control lines
36--42	H	LS, FAA Read/Write controls and LS addressing.
43--46	L	Storage requests and setting of mark triggers
47--56	NA	Base address of next ROS word
57--61	K	Y-conditional branches
62--68	J	Z- (and X-) conditional branches
69--73	M	Serial adder A bus
74--77	N	Serial adder B bus
78--80	P	Parallel adder latches
81	*	Blocks normal gates to serial adder and gates N-register
82--84	Q	Hot 1's to parallel adder A-side
85	–	Parity
86	R	F- and AB-register outgating to serial adder A-bus
87--90	T	A-, B-, IC-, and F-register outgating to parallel adder B-bus
91	–	Parity
92--96	U	S-, T-, and D-register outgating to parallel adder A-bus, mixer controls
97--99	V	E- and Q-register outgating to parallel adder B-bus

† Control fields D, F, and G serve two functions. In the normal processing mode, they are decoded to yield standard CE micro-orders; in the scan mode, they are identified as field S, and they yield special scan micro-orders (using common micro-order codes). The choice of modes is controlled by a 'scan mode' trigger.

* Bit 81 is the high-order bit of the E-field.

Each ROS word is represented by a block on a Control Automation System (CAS) Logic Diagram (CLD). The CLD is to the ROS microprogram what an ALD (Automated Logic Diagram) is to logic. The blocks are connected to show the logical sequence of ROS words to perform the specific function. Refer to ALD A6281 for a definition of the CLD format and content, and of the ROS block language and information contained within the block.

ROS Addressing and Branching

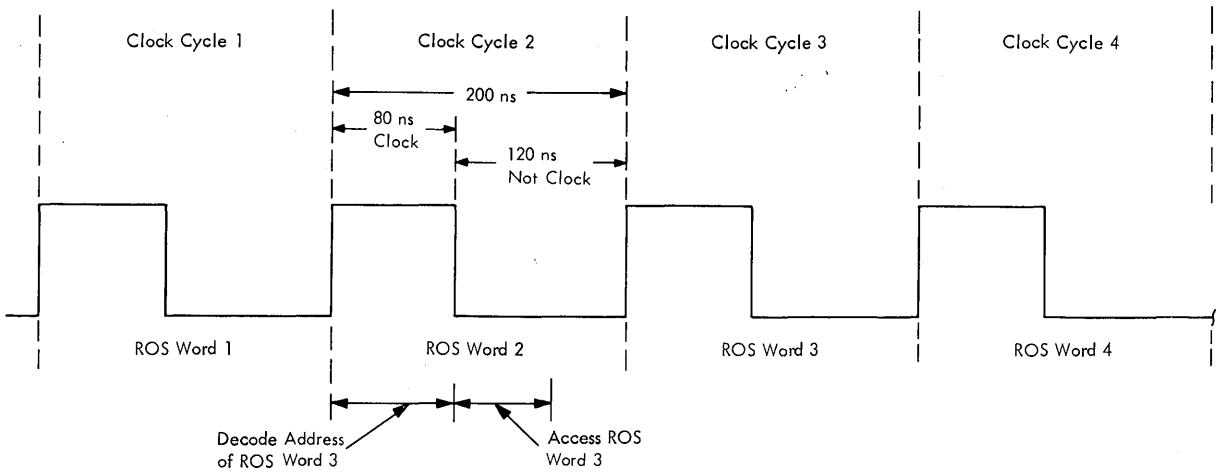
- Conditional branches are dependent on internal conditions of previous cycle.
- Word addressed as result of branch test is not available until 1 cycle later.
- ROS word is addressed by 12-bit binary address:
  0–9 is 10-bit base address.
  10 is Y-branch bit.
  11 is Z-branch bit.
- X-branch (functional branch) affects more than 1 bit.
- Overriding branch blocks 12-bit address and forces new 12-bit address into ROSAR.

As described earlier, the machine cycle presently being executed is controlled by the ROS word addressed during the previous cycle. Referring to Figure 1-4, A, the normal sequence of ROS words is achieved by placing the address of ROS word 2 into ROS word 1, the address of ROS word 3 into ROS word 2, and so on. The address of the next ROS word is decoded during clock time of the cycle controlled by the present ROS word; the next ROS word is accessed during not-clock time of the cycle.

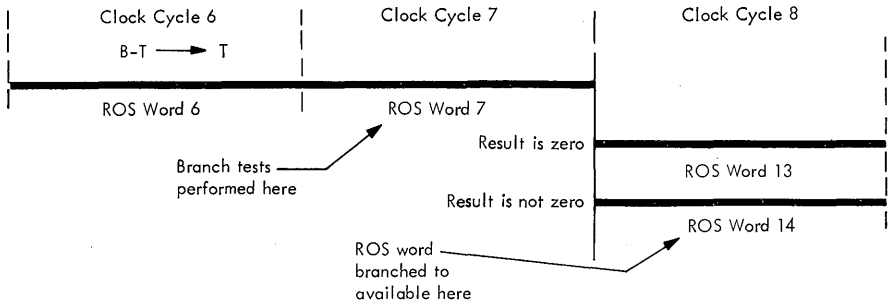
Conditional branches are dependent on the internal conditions of the previous cycle. It is important to note that the ROS word addressed as a result of the branch test is not available until one cycle later. To explain this 1-cycle delay in addressing, assume that ROS word 6 contains the micro-orders necessary to subtract the contents of the T-register from the contents of the B-register and to place the result into the T-register (Figure 1-4, B). Assume further that if the result is zero, a branch will be made to ROS word 13; if not zero, the next ROS word addressed will be 14. Contained in ROS word 6 is the address of ROS word 7, which defines the branch test and contains the associated branch addresses.

The results of the arithmetic operation performed in cycle 6 are tested during clock time of cycle 7. It is during this time that the address of ROS word 13 or 14 (depending on the results of the branch test) is decoded: the selected ROS word is accessed during not-clock time of cycle 7. Hence, the ROS word branched to as a result of the arithmetic operation performed during cycle 6 is not available until cycle 8.

ROS words are selected by means of a 12-bit binary address. The address is held in the ROS address register (ROSAR), whose bit positions are numbered 0 through 11, high order to low order. Bits 0 through 10 specify a 200-bit doubleword in ROS; bit 11 gates the proper 100-bit half to the ROS sense latches (Figure 1-5).



A. Sequential ROS Word Addressing



B. Conditional Branch ROS Word Addressing

Figure 1-4. ROS Addressing and Branching

The 12-bit address is made up of three components:

1. A 10-bit base address, bits 0–9.
2. A conditional branch test, or an unconditional value of 0 or 1 applying to bit 10, designated Y-branch.
3. A conditional branch test, or an unconditional value of 0 or 1 applying to bit 11, designated Z-branch.

Included in the Z-branch field of micro-orders is a subset of branch micro-orders called X-branch or functional-branch micro-orders. The X-branch micro-orders affect more than one bit of the ROS address.

Included in the Y-branch field of micro-orders is a subset of overriding branch micro-orders. When the conditons tested by these micro-orders are satisfied, the full 12-bit address is blocked and a new 12-bit address is forced into ROSAR.

If branching conditions are to be tested, the address bits that may be affected by the branch must be set to 0's, except in the case of an overriding branch. If the branch is satisfied, 1's are forced into the ROSAR bit positions associated with that branch test; if the branch condition is not satisfied, the bits remain 0's. Thus the address is modified only if the branch is to be taken.

Addresses can be grouped into four categories: (1) no branch specified, (2) Y- and/or Z-branches, (3) X-branches, and (4) overriding branches. The following paragraphs discuss the addressing for each category. Refer to ALD A6281 for a definition of ROS addressing and branching terms used in the following paragraphs.

No Branch Specified

If no branch tests are to be made, there is only one possible ROS word that can follow, and hence only one possible next address. Accordingly, the 10-bit base address (bits

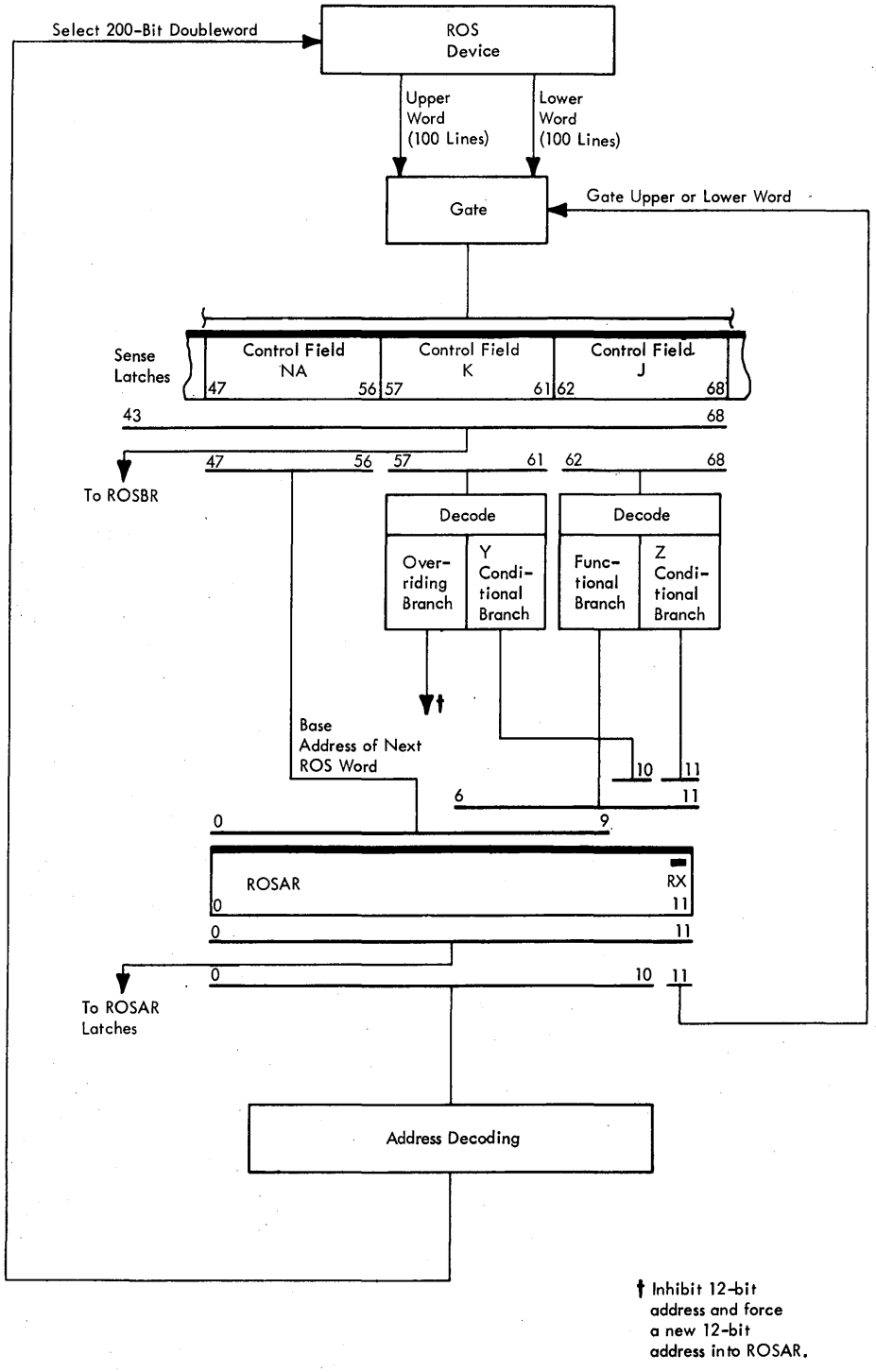


Figure 1-5. ROS Addressing Block Diagram

0–9) and absolute values of 1 or 0 for bits 10 and 11 are specified. The micro-orders that unconditionally set an absolute value into bits 10 and 11 are:

1. 0 in left of R-line (K0), which sets bit 10 to a 0.
2. 1 in left of R-line (K1), which sets bit 10 to a 1.
3. 0 in right of R-line (J0), which sets bit 11 to a 0.
4. 1 in right of R-line (J1), which sets bit 11 to a 1.

The appropriate Y- and Z-branch micro-orders are selected, and bits 10 and 11 are set correspondingly.

Y- and/or Z-Branches

Conditional branch addresses may be specified in which bits 10 and/or 11 are affected.

Only a Y-branch can be executed as follows. A 10-bit base address and an absolute value or X for bit 11 are specified. A branch test is defined in the Y-branch micro-order control field. If the branch condition is satisfied, bit 10 is set to a 1; if not, bit 10 remains a 0. For example, micro-order 'WCRY' sets bit 10 to a 1 if a carry is detected in the serial adder. If there is no carry, bit 10 remains a 0.

Conversely, only a Z-branch can be executed as follows. Here, a 10-bit base address and an absolute value for bit 10 are specified. If the branch test defined in the Z-branch micro-order control field is satisfied, bit 11 is set to a 1; if not, bit 11 remains a 0.

Certain situations require the use of both Y- and Z-conditional branches simultaneously. The 10-bit base address is specified, and bits 10 and 11 may assume one of the following values:

Bits		Branch Results
10	11	
0	0	Y- and Z-branch conditions both unsatisfied.
0	1	Z-branch condition only satisfied.
1	0	Y-branch condition only satisfied.
1	1	Y- and Z-branch conditions both satisfied.

X-Branches

Where a branch to one of four or more possible addresses is required (as well as some special 64-way branch tests), an X-branch is used. The X-branch may affect bits 10 and 11 (four-way branch), bits 9–11 (eight-way branch), bits 8–11 (16-way branch), or bits 6–11 (64-way branch). An example of an X-branch is the 64-way branch, 'E(2–7)→ROA', made at the end of the I-Fetch sequence per the op code to enter the execution phase of the specific instruction.

For these multiway branches, one condition sets the associated address bit to a 1. To illustrate, assume conditions A, B, and C sets bits 9, 10, and 11, respectively, to a 1. The possible results are:

Bits			Branch Results
9	10	11	
0	0	0	None of the conditions is satisfied.
0	0	1	Condition C is satisfied.
0	1	0	Condition B is satisfied.
0	1	1	Condition B and C are satisfied.
1	0	0	Condition A is satisfied.
1	0	1	Conditions A and C are satisfied.
1	1	0	Conditions A and B are satisfied.
1	1	1	All three conditions are satisfied.

The addressing is similar to that previously discussed. A 10-bit base address is specified, with those bits that may be affected by the X-branch set to 0. Thus, for the example given above, bit 9 of the base address is set to 0, the Y-branch micro-order control field contains micro-order 0 in left of R-line to set bit 10 to 0, and bit 11 is automatically set to 0 when the X-branch is specified. Subsequently, the bit(s) associated with successful condition(s) is set to 1. The ROS word addressed will be that ROS word whose address satisfies the branch conditions.

Overriding Branches

There are exceptional machine conditions (such as interruptions) for which the normal ROS word sequence must be stopped and a new sequence started. This change is accomplished by an overriding branch specified in the Y-branch micro-order control field.

The normal sequencing address is made up of (1) the 10-bit base address and (2) bit 11 set to 0 automatically because the overriding branch is a function of the Y-field. Because the overriding branch is specified in the Y-control field, no Y-branch can be specified.

If the overriding branch condition is satisfied, the normal full 12-bit address is blocked and a new address, as determined by the overriding branch condition, is forced into ROSAR.

ROS Data Flow

- ROS data flow units are:
  100 sense latches, 1 per ROS word bit
  ROSAR
  ROSAR latches
  PROSAR A and PROSAR B
  ROSDR
  ROSDR latches
  ROSBR
  Decode logic

- Control fields may be:
  Decoded directly from sense latches.
  Placed into ROSDR and subsequently decoded.
  Placed into ROSDR, sent to ROSDR latches, and then decoded.

The 100-bit ROS word is divided into 22 control fields. When read out from ROS, the ROS word is placed into 100 sense latches, one latch for each bit position. The control fields are handled according to the functions they control. They may be (Figure 1-6):

1. Decoded directly from the sense latches (control fields L, K, J, R, T, U, and V) or transferred directly to ROSAR (control field NA).
2. Placed into the ROS data register (ROSDR) and decoded (control fields H, M, N, P, and Q).
3. Transferred to ROSDR latches from ROSDR (control fields A–G and W).

Assuming ROS words and the cycles they control are designated 1, 2, and 3, ROS word 1 is set into the sense latches during not-clock time of cycle 0 (Figure 1-7). The control fields used during clock time of cycle 1 are decoded directly from the sense latches (case 1 above). These control fields, which may be considered critical timing fields, control inputs to the adders, define the base address and branch tests for the next ROS word, and control the storage-request and mark triggers.

Although the sense latches are cleared at not-clock time of cycle 1, the control fields of ROS word 1 that are required during that time (case 2 above) are placed into ROSDR at clock time of cycle 1. These signals control the adders and LS. Note that both portions of the ROSDR associated with the adders are packaged physically with the adders they control. The balance of the ROSDR serves control fields A–H and W.

Control fields A–G and W control register inputs and triggers that are to be set during clock time of cycle 2 (case 3 above). Although the ROSDR is reset at the end of cycle 1, the ROSDR latches keep control fields A–G and W available for that additional 80 ns (Figure 1-7).

Control fields L, NA, K, J, R, T, U, and V are sent to the ROS backup register (ROSBR) from the sense latches. The ROSBR does not play a part in the ROS functions: it provides an indication of the subject fields during maintenance (Test) mode. When the CE stops on an error during test mode, the ROSBR contents can be used by maintenance personnel to help isolate malfunctions.

The NA control field, in addition to being stored in the ROSBR, is stored in ROSAR and provides the base address as previously explained. During each ROS cycle, the contents of ROSAR are sent to the ROSAR latches which, in turn, are alternately gated (by means of an 'A-gate' signal) to the previous ROS address register A (PROSAR A) and the previous ROS address register B (PROSAR B). These registers serve the same purpose as the ROSBR, i.e., provide an indication for maintenance use during Test mode. When an error causes the CE to stop in Test mode, ROSAR, PROSAR A and PROSAR B provide the addresses of the next ROS word, current ROS word, and previous ROS word (Figure 1-7).

ROS Control of CE

Efficient control of CE operations is achieved by overlapping ROS words. Clock signals (P0–P19) time ROS sense latches, ROSDR, and ROSDR latches, thus allowing the processing of parts of more than one ROS word simultaneously. To illustrate, Figure 1-8 shows the ROS word timing relationship for a hypothetical example.

The address of word 1 is gated into ROSAR from the ROS sense latches at P5 of word 0. [ROSAR(11) may be set as late as P7.] Information from word 1 enters the ROS sense latches at P0 + 160 ns (P16). (In normal operation, a new word enters the ROS sense latches every 200 ns.) In the example, word 1 controls: (1) register output, (2) main storage request, (3) adder shift, (4) local storage write, and (5) register input. A register output micro-order gates register data into an adder at P2 by means of 'gate control' triggers. Note that register output and the main storage request are decoded directly from the ROS sense latches. A three-cycle main storage request is initiated at P4 to fetch information which is used three cycles later. The clock is stopped as shown in Figure 1-8 to account for cable delays and storage busy delays. The 'accept' signal from storage is used to turn the CE clock back on in order to sync it with data on the SDBO.

Bits 2–42 enter ROSDR at P0, and bits 69–80 and 82–84 enter ROSDR at P2; they are decoded for adder control and LS operations. In the example, micro-orders generate an adder shift and a local storage write operation. The shift is performed immediately, but the local storage write operation is delayed because a local storage read operation is automatically set up first. The LS address is entered into the local storage address latches (LAL); a read operation is performed, but data is not gated into a register.

The address then is gated into the local storage address register (LAR) to perform the write operation. Note that the write condition, ordered by word 1, starts after word 2 has been transferred to the ROSDR. The figure shows an LS read operation for every word because this sequence happens even if it is not ordered. When no order is given to LS, a readout of LS address 0 is performed but the data is not used. Forcing the read operation saves time if it is needed.

ROSDR(2–35) is gated to the ROSDR latches at P7 to retain word 1 information at the same time word 2 is set into ROSDR. In the example, word 1 transfers data into a register at clock time (P2) of word 2. This action, along

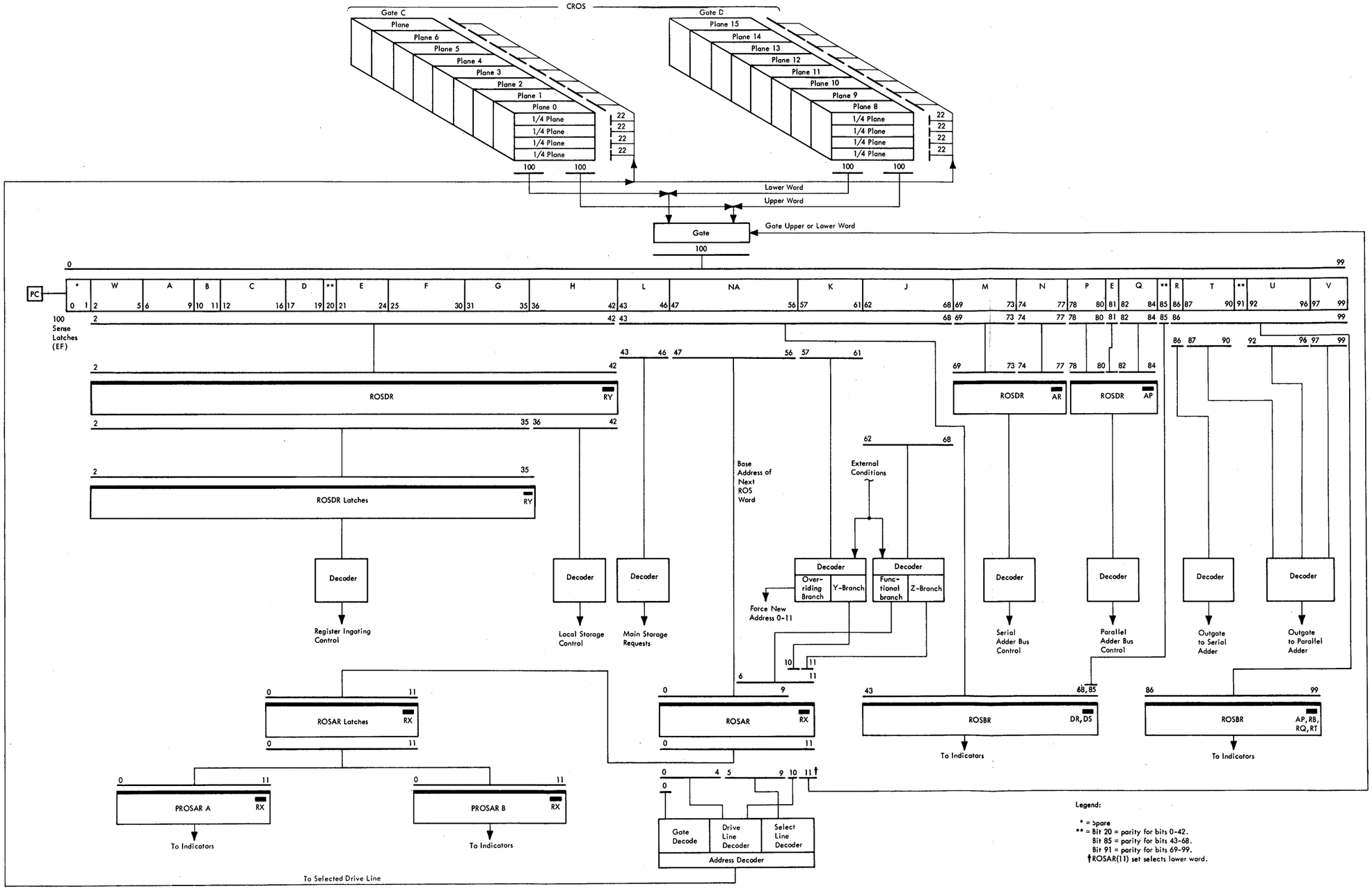


Figure 1-6. ROS Data Flow

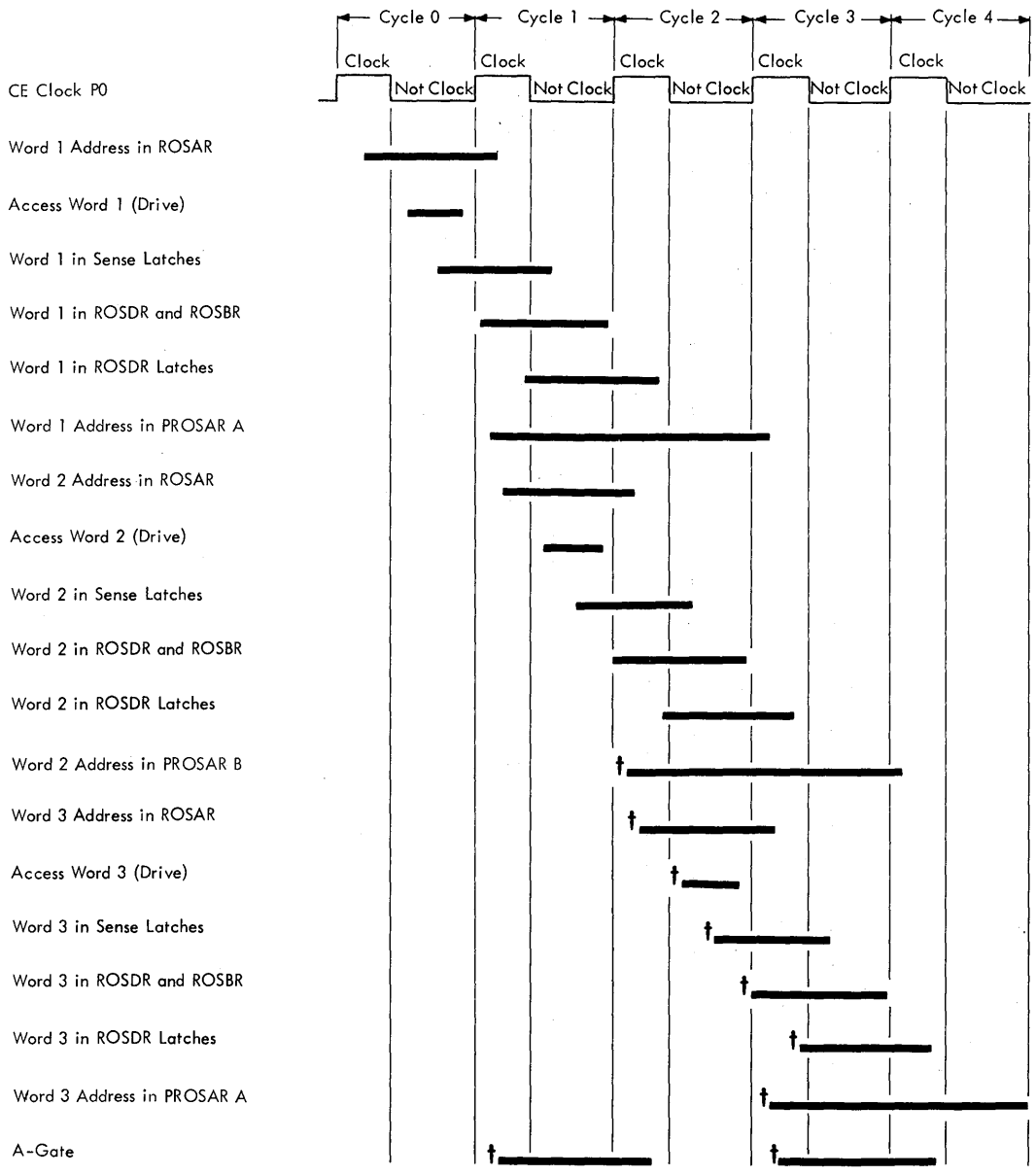


Figure 1-7. ROS Timing

with an LS write operation, illustrates ROS word overlap because these operations are performed at the same time the register out condition is energized from the decoding of word 2.

Word 2 has only one micro-order, register output, but gates are conditioned to transfer word 2 data from the ROS sense latches to the ROSDR and then to the ROSDR latches. Note that the LS read micro-order is active, though not ordered. Word 3 operates in a similar manner, but the only micro-order is a register input which takes place during word 4.

As the ROS words are executed, the main storage request is processed and data is returned on the Storage Data Bus Out (SDBO); word 4 contains the micro-order to gate the data into a register for further processing.

PSW REGISTER

Program status words (PSW's) contain detailed information pertaining to the particular mode in which the CE is operating. These status words are composed of a system

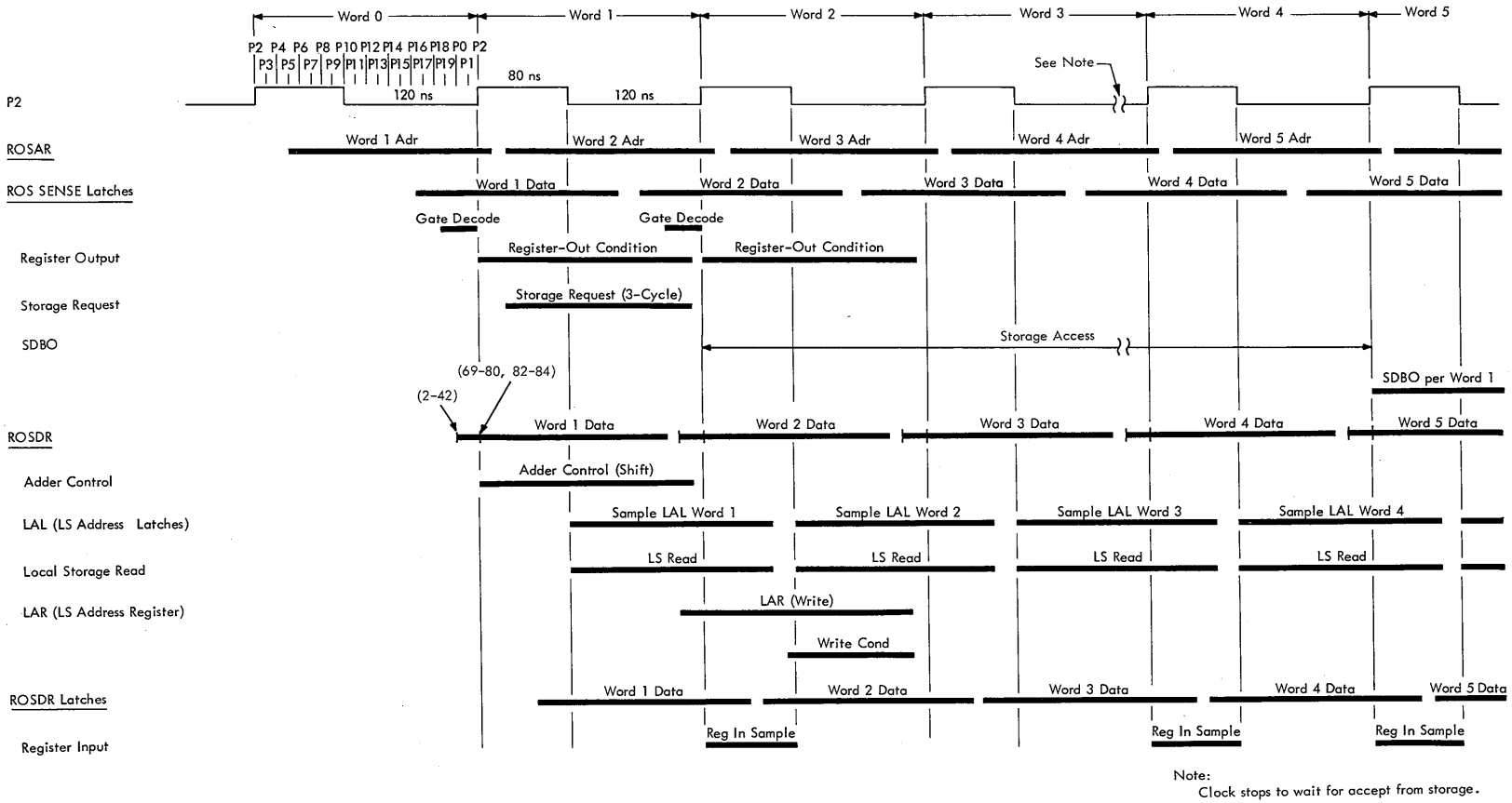


Figure 1-8. ROS Control of CE Operations

mask, storage key, program state, interruption code, instruction length code, condition code, program mask, and an instruction address that enables the interrupted program to resume at the correct location.

Status information concerning the current operating program is contained in several groups of triggers, from where it controls all system operations essential to that particular program mode. These groupings of control triggers, although not adjacently located in logic, are collectively referred to as the PSW register. Although a completely assembled PSW is 64 bits long, only 28 positions of status word data are contained in the PSW register. The remaining information (generated by the CE at the time of the interruption) is not retained when a previously stored PSW is reloaded, because its function is only to identify the cause of the interruption and to return the CE to the correct program location. (This information is gated directly from ST when the old PSW is recalled from main storage.) Figure 1-9 shows an assembled PSW and those areas of control information retained in the PSW register.

CONFIGURATION CONTROL

The 9020 systems have the facility to alter system configuration under program control. The reconfiguration capability is made possible by the existence of redundant elements and the extensive interfacing between system elements. These interfaces are gated by configuration control registers (CCRs). The CE, like other major system elements, contains a CCR. This CCR determines three conditions for the CE: (1) other elements with which it may communicate, (2) element state it is to assume, and (3) from which CEs it may accept a reconfiguration.

The CE has the capability of setting the CCRs in any element, including itself. Therefore, hardware exists in the CE for both sending and receiving configuration data. CEs send configuration data to other elements (or to themselves) by executing the Set Configuration (SCON) instruction.

Special interface lines exist to most elements to transfer the configuration data. In the case of the SEs and the DEs, configuration data is sent over the Storage Data Bus In (SDBI).

In the 9020E system, RCUs extend the reconfiguration capability to the DAUs and certain display equipment, but the RCUs do not use the SCON instruction to do so. The RCUs do have to be configured via a SCON instruction, however.

Figure 1-10 shows the simplified data path for configuration control. Three registers are used which have not been discussed so far: the CCR, the external register, and the select register.

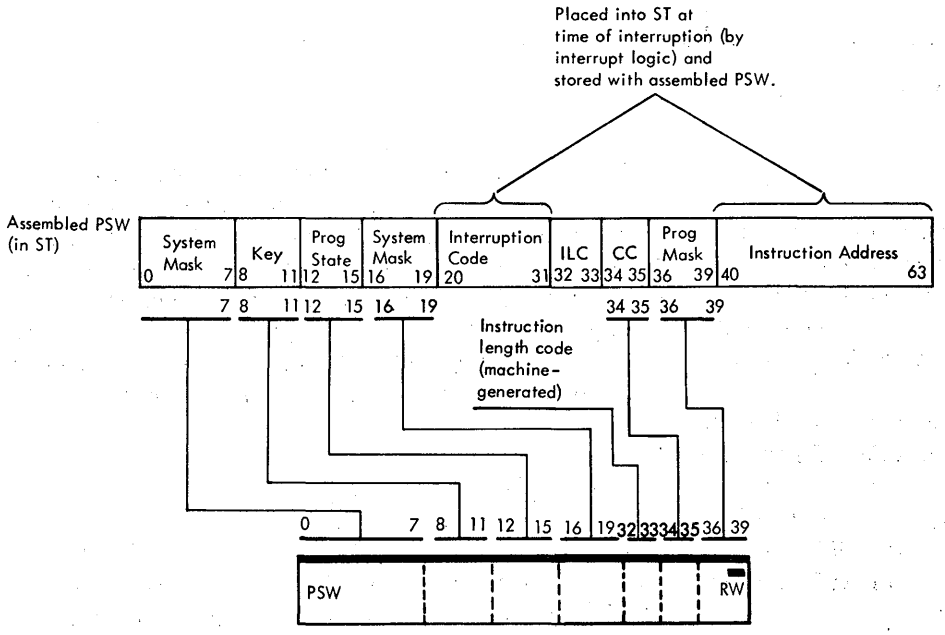
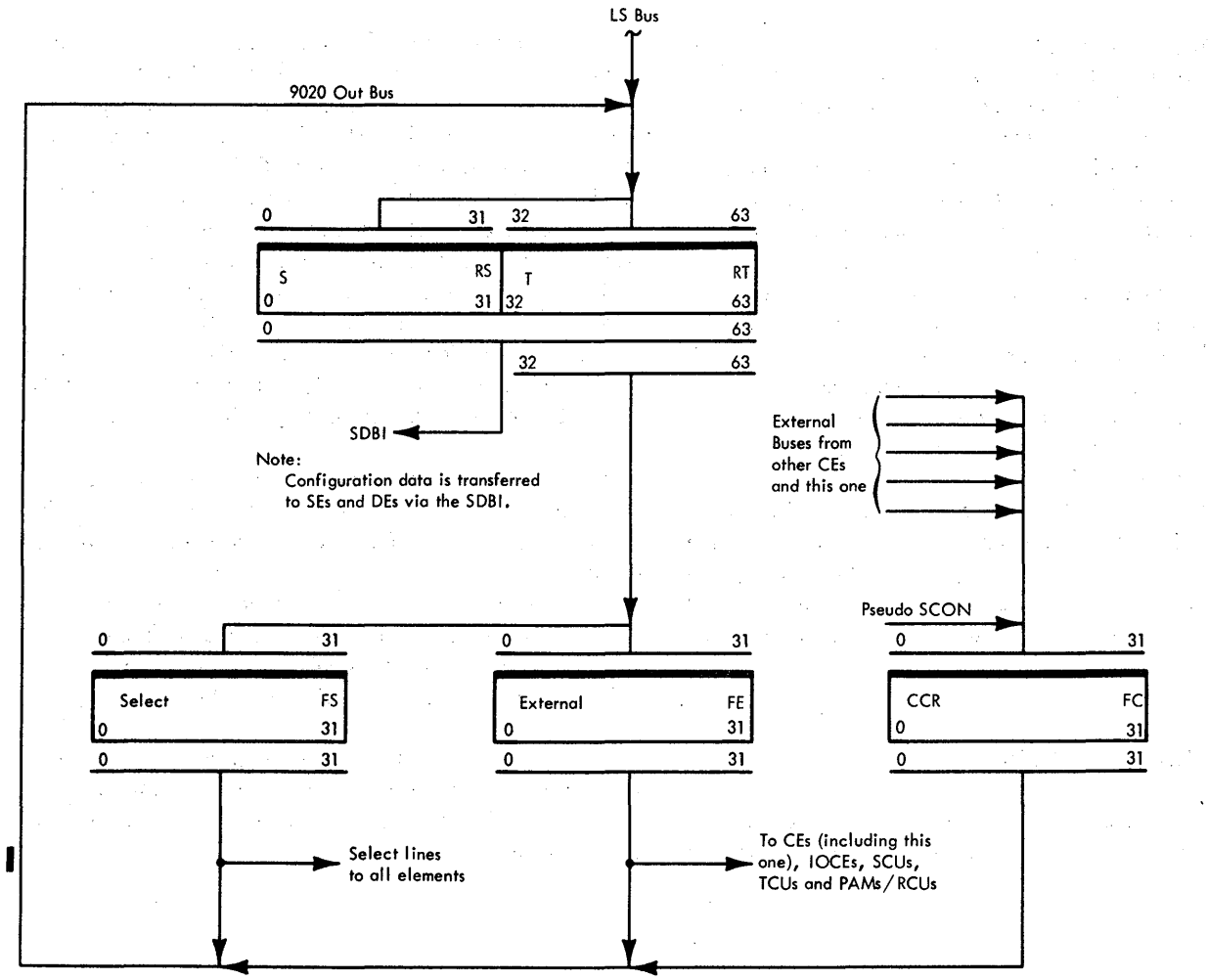


Figure 1-9. Status Information Contained in PSW Register



●Figure 1-10. Configuration Control Data Paths

Configuration Control Register (CCR)

The CCR in the CE is a 32-bit register consisting of a state field, a SCON field, and a communications field. Additionally, an inhibit logout-stop (ILOS) bit is contained in the CCR. An explanation of these fields is given in the system introduction manuals, together with an extensive introduction to configuration control.

The CCR is set from the external buses of each CE in the system. (Refer to Figure 1-10.) A particular CE's external bus also wraps back to itself so that it can set its own CCR. One additional input, pseudo SCON, is used by the CE to force a configuration of itself during system IPL or system PSW restart. The CCR can be gated to the 9020 Out Bus and into S or T via the Local Store Bus.

External Register

The External bus of each CE originates in that CE's external register. There is no further gating, so that data in a CE's external register is effectively on the external Bus.

The external register is a 32-bit register which provides for the storage of data to be routed to CEs, IOCEs, TCUs, SCUs, and PAMs or RCUs. Between CEs and between CEs and IOCEs, the external bus is used for other purposes in addition to the transmission of configuration data. For this reason, it is often called the Control Bus. The external register may also be gated onto the 9020 Out Bus.

Select Register

The select register is a 32-bit register whose bit settings define the elements to be selected during execution of a SCON instruction. Lines from the bit positions carry the 'select' signal to the corresponding elements. These 'select' signals are then used by the selected elements to gate the configuration data into their CCRs.

The select register is also used to select CEs and IOCEs during a Set Address Translator (SATR) instruction, as is explained later. The output of the select register may also

be gated onto the 9020 Out Bus for transfer to S or T. This is necessary because the bits of the select register are normally reset by responses from the individual elements selected by the SCON instruction and a path is needed to enable the microprogram to examine the select register to determine whether all elements responded. This is discussed further later in this chapter when the SCON and SATR instructions are introduced.

STORAGE ADDRESSING

Main storage in the 9020 systems is external to the CEs being provided by stand-alone storage elements (SEs). The SEs use two-way data interleaving to achieve a faster effective access time. This is accomplished by storing consecutive doublewords in separate basic storage modules (BSM) within each SE. Even-numbered doublewords are stored in an even BSM; odd-numbered doublewords are stored in an odd BSM. Since buses and SE common circuitry are required only during the read portion of a storage cycle, a second storage cycle can be initiated for the other BSM while the write half of the cycle for the first BSM is being completed. This represents a large saving of time, especially when a series of consecutive doublewords are being accessed.

Provision is made to defeat this interleaving for maintenance purposes, so that a program can be loaded in one BSM or the other should one BSM be failing.

Storage is addressed by the storage control interface (SCI) portion of a CE based on the program-established setting of certain registers. These registers are the address translation register (ATR) and the preferential storage base address register (PSBAR). These registers enable the reconfiguration of SEs and DEs and make possible the programmed establishment of PSAs. The SCI is discussed after the description of these registers and their interrelationship.

Address Translation Register (ATR)

The ATR comprises two registers: a 32-bit (plus four parity bits) register (ATR 1), and an eight-bit (plus parity) register (ATR 2). The two together may be logically considered as one 40-bit register. This 40-bit register is divided into ten four-bit sections called slots. Bits 9–12 of a logical address are used to select an ATR slot. The selected slot contains an SE identifier, placed in the slot by programming, which is substituted for bits 9–12 of the logical address to produce the translated address. The program uses the Set Address Translator (SATR) instruction to initialize the ATR.

Preferential Storage Base Address Register (PSBAR)

PSBAR also comprises two registers (called physical PSBAR and logical PSBAR). Logical PSBAR is a 12-bit register which can be set by execution of the Load Preferential Storage Base Address (LPSB) instruction. It contains bits 8–19 of the address of the PSA. These bits are sufficient to define the PSA location because the PSA is a 4096-byte area of storage. The missing low-order bits would define a location within the 4096-byte area specified by PSBAR.

Physical PSBAR is a four-bit register which contains the SE identifier representing the ATR-translated value of bits 9–12 from logical PSBAR. Physical PSBAR is translated via the ATR and is set during execution of the LPSB instruction.

Through programming convention, the PSA is accessed by addressing the lowest 4096-byte area of available storage. These addresses are handled by PSBAR in such a manner as to access the actual location of the PSA. All other accesses from the program are called "normal accesses" and are handled by the ATR. The two types of accesses are differentiated by the CE through examination of the 12 high-order bits of the address. An all-0's condition indicates a PSA access.

Figure 1-11, in simplified form, shows the relationship between the registers used for normal and PSA storage accesses. The ATR is set from S(0–31) and T(32–39) during the SATR instruction execution. Outputs are provided to the 9020 Out Bus for examining the ATR during execution of the Insert Address Translator (IATR) instruction. Each slot can be gated independently to the storage address bus (SAB) and storage selection logic located in the SCI.

An ATR slot decoder controlled by bits 9–12 of logical PSBAR enable the translation of those bits into the proper identifier to be set into physical PSBAR. During PSA accesses, physical PSBAR, bits 13–19 of logical PSBAR, and the low-order bits of the original address are ORed to produce the actual PSA address for storage accessing. These bits from PSBAR are shown in Figure 1-11 being sent to the SAB and storage selection logic for this purpose. Data paths are also available from both logical and physical PSBAR to the 9020 Out Bus so that their contents may be stored for program examination through execution of a Store Preferential Storage Base Address (SPSB) instruction (which is described later).

The PSBAR counter shown in Figure 1-11 makes possible the automatic stepping of PSBAR when a CE is unable to access the original PSA. In the event of such a malfunction, the gate to bits 9–12 of logical PSBAR causes the output of the PSBAR counter to be gated in. The

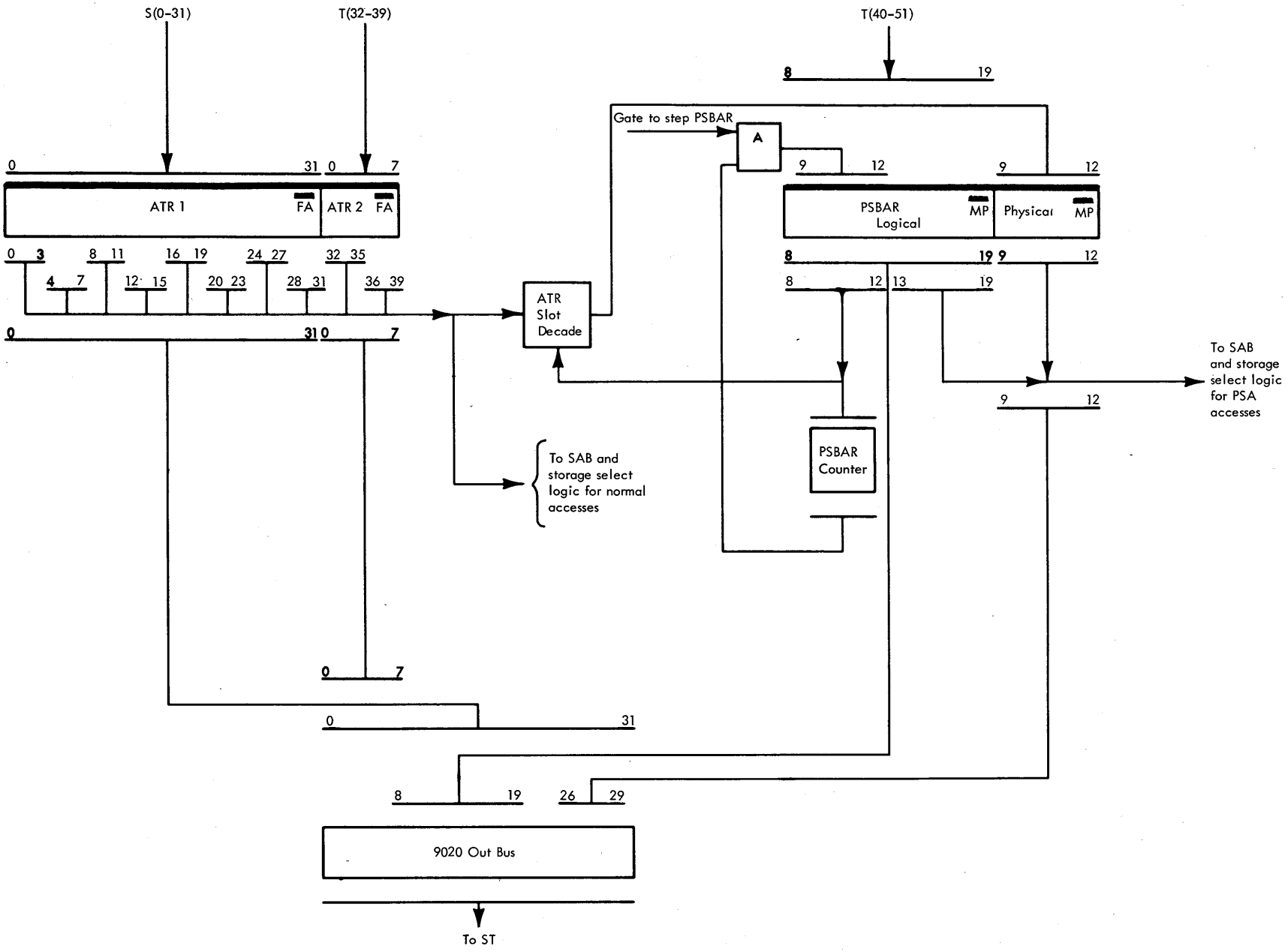


Figure 1-11. Storage Addressing Flow, Simplified

PSBAR counter is always incremented by 1 so that the next-higher ATR slot is set into logical PSBAR. When the next configured SE is found, the value from the new ATR slot is set into physical PSBAR, and a new PSA access is initiated.

The SCI portion of the CE controls storage accessing and the stepping of PSBAR.

STORAGE CONTROL INTERFACE (SCI)

Storage control interface (SCI) is a major section of the CE and contains all necessary logic for handling of CE storage requests. When a storage request is issued by the CE, SCI gates an address from D, IC, or PAL (according to the type of request) and issues a 'select' pulse to the proper SE or DE. SCI logic will stop the clock at the end of the cycle following a select pulse cycle and will restart it after receiving an 'accept' signal from the SE or DE.

Major Interface Lines

The CE to SE and DE interface lines are classified by three terms, as defined below.

1. Distributed simplex. Bus and control lines that are common to all SEs and DEs in a system and carry signals from a CE, e.g., SAB(19+1P).
2. Multiple driver distributed simplex. Bus and control lines that are common to all SEs and DEs in a system and carry signals to a CE, e.g., SDBO (64 + 8P).
3. Simplex. One-way control lines that carry signals between a CE and each SE or DE individually, e.g., SAB P(A) SE1.

The interface lines shown in Figure 1-12 are defined as follows:

1. Storage Address Bus (SAB). This distributed simplex bus carries the signals for a 19-bit address with 1 parity bit (for SAB 1–5) to all SEs and DEs in a system.
2. Storage Address Bus Parity – SAB P(A) and SAB P(B). These two groups of simplex lines carry the parity bit signals for SAB(6–12) and SAB(13–19), respectively.
3. Storage Data Bus Out (SDBO). This multiple driver simplex bus carries the storage data and parity signals from all SEs and DEs in a system to a CE. SDBO consists of 64 data and 8 parity lines.
4. Storage Data Bus In (SDBI). This distributed simplex bus carries the storage data and parity signals from a CE to all SEs and DEs in a system. SDBI consists of 64 data and 8 parity lines.
5. Mark Bus. This distributed simplex bus carries signals that designate which bytes on the SDBI are to be stored into main storage. A mark line corresponds to each byte of the doubleword. If a mark signal is active, the byte on SDBI that corresponds will be stored; if the mark signal is not active, data from storage will be restored by the SE or DE. Mark bus has eight mark lines and one parity line.
6. In Key Bus. This distributed simplex bus carries storage protection key signals from a CE during fetch and store data cycles and during store key cycles. It has five key lines and one parity line.
7. Out Key Bus. This multiple driver simplex bus carries storage protection key signals to a CE during insert key storage cycles. It has five key lines and one parity line.
8. Select Odd and Select Even. These two groups of simplex control lines carry 'select' signals from a CE to each SE and DE in a system (one 'select odd' and one 'select even' line to each SE and DE). A 'select' signal will cause a SE or DE to establish priority for the CE and start a storage cycle when priority is granted.
9. Accept. This group of simplex control lines carries signals to indicate to a CE that a storage cycle has been started as a result of a 'select' signal from the CE.

Storage requests are generated in the CE and sent to the SCI to develop a 'select' signal to an SE or DE. The signals that perform this function are related to the source of the storage address as follows:

1. D storage request. The storage address is in the D-register.
2. IC storage request. The storage address is in the instruction counter.
3. Scan storage request. The storage address is developed in scan logic.

The signals developed by the SCI to control the buses are:

1. 'Gate IC to SAB'. Gates the contents of the instruction counter to SAB.
2. 'Gate D to SAB'. Gates the contents of the ST register to SDBI.
3. 'Gate scan to SAB'. Gates the address developed by the scan logic to SAB.
4. 'Gate F to key-in'. Gates the contents of the F-register to the 'key in' bus.
5. 'Gate ST to SDBI'. Gates the contents of the ST register to SDBI.
6. 'Gate XY to SDBI'. Gates the contents of the XY register to SDBI.
7. 'Stop CPU clock'. Stops the clock when a 'select' signal is sent to an SE or DE.
8. 'Gate SDBO to AB'. Gates the data on SDBO into the AB register.
9. 'Gate SDBO to LM'. Gates the data on SDBO into the LM register.
10. 'Gate SDBO to Q'. Gates SDBO into the Q-register.

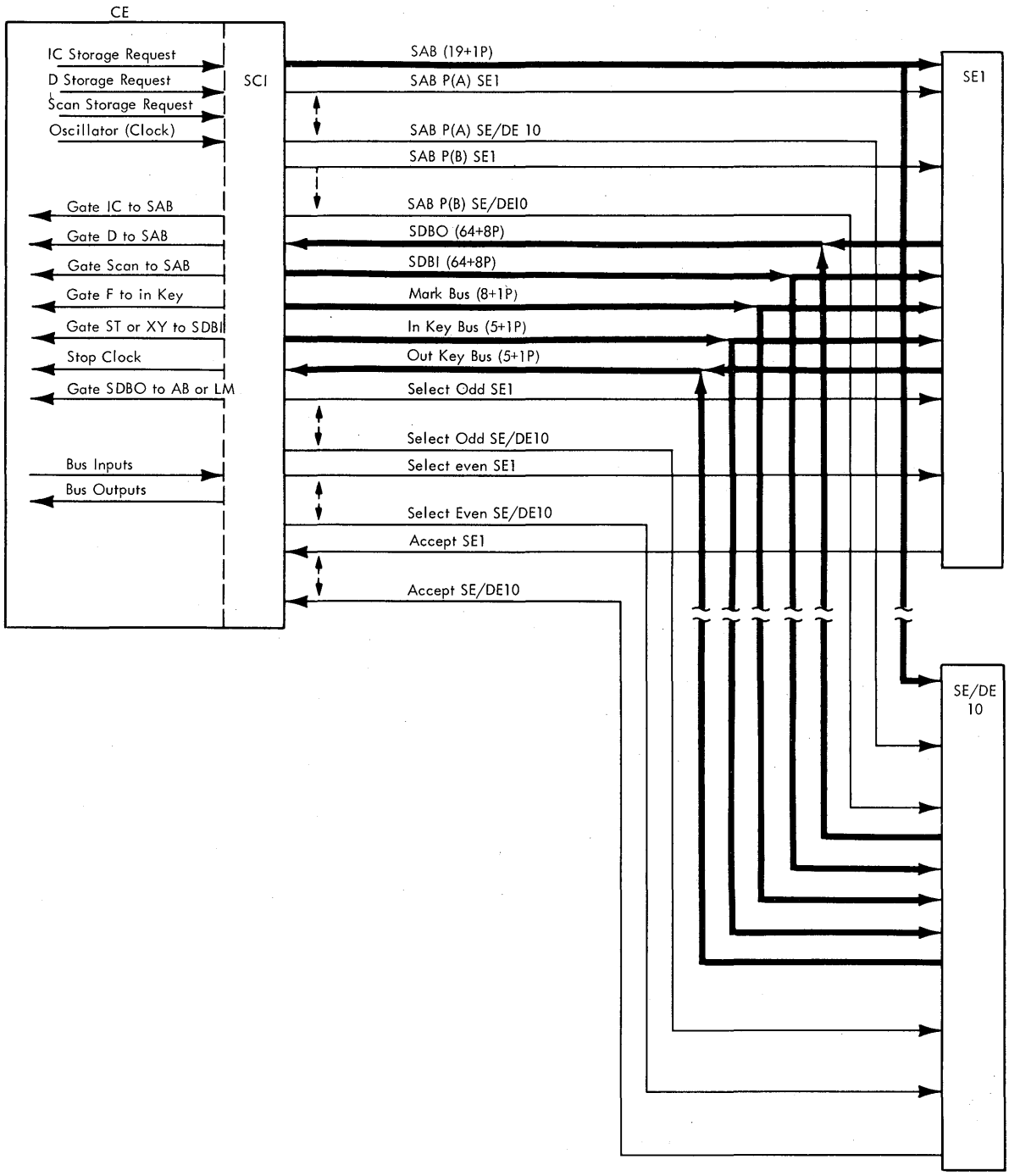


Figure 1-12. Basic CE to SE/DE Interface

CE Storage Requests

- Address is decoded to select SE or DE.
- Invalid address is detected if selected SE or DE is stopped or not ready.
- Step PSBAR to alternate SE.

For the following discussion, the reader must be aware of two numbering schemes used to identify main storage address bits. The D- and IC-registers (sources of main storage address) are displayed on the CE panel as bits 8–31; these bits are designated as 0–23 in the ALDs. To be consistant with the ALDs, this discussion references the address bits and related SCI logic as 0–23. The following listing shows how the two numbering schemes relate to each other.

CE Console Labeling For D and IC Register Display	CE Logic Line Labeling For D and IC Registers, and All SCI Logic
8	0
9	1
10	2
11	3
12	4
13	5
14	6
15	7
16	8
17	9
18	10
19	11
20	12
21	13
22	14
23	15
24	16
25	17
26	18
27	19
28	20
29	21
30	22
31	23

SCI contains logic for handling storage requests from the CE. A storage request is generated from one of three sources: (1) instruction fetch logic to refill the instruction buffer (Q-register), (2) microprogram to fetch or store data, and (3) scan logic.

The address of the desired doubleword in main storage is located in D- or IC-register or is generated by the scan logic. SCI logic decodes the address to activate a 'select' pulse to the SE or DE.

The SE or DE is selected in one of two ways: (1) if the address supplied by the CE contains all 0's in bits 0–11 (preferential storage area address), the contents of physical perferential storage base address register (physical PSBAR) are decoded, or (2) if address bits 0–11 are not all 0's, bits 1–4 of the address are decoded to select four bits of the address translation register (ATR slot), which are decoded.

An invalid address condition will be detected in SCI logic if the selected SE or DE is not ready (power down, state 0 with test switch in TEST position, or not configured to the CE) or is stopped for a logout. This condition will cause a program interrupt.

If there is no invalid address condition, a select pulse is sent to the decoded SE or DE. The clock is stopped at the end of the cycle following a select cycle and restarted after an 'accept' pulse is received from the SE or DE.

SCI logic develops the gating signals for SAB, SDBO, SDBI, 'mark' bus, and the storage address protection keys. It checks for errors and handles error signals from SEs and DEs. If an invalid address condition or a storage error is detected during a PSA access, SCI logic will "step PSBAR" to relocate the PSA in an alternate SE.

Page Controls

- Page of data on 512-byte bounds.
- Page controls provide linkage to page overflow address.

A page of data is a maximum of 512 bytes stored on 512 byte bounds; it may be overflowed into another page. Page controls in SCI logic provide a means of linkage to a page overflow address when fetching (radar) data.

The page overflow address will be in bits 40–60 of the last doubleword in a page. When fetching radar data, the microprogram tests for a PAL 512 carry as the storage is incremented by 8. When a 512 carry is detected, page controls block normal resetting of SCI logic, which forces an additional storage fetch cycle to the next sequential doubleword. Page control logic gates SDBO (40–60) to D or IC, according to the type of request, and the page operation is complete.

INSTRUCTION FETCHING

The processing of an instruction is divided into two phases: instruction fetching and execution. Instruction fetching, or I-Fetch, retrieves instructions from main storage and performs operations common to many instructions. For the most part, I-Fetch, which is controlled by ROS and conventional hardware:

1. Determines the address to be placed into the instruction counter (IC).

2. Fetches instructions from main storage.
3. Determines the instruction format (RR, RX, RS, SI, or SS).
4. Calculates the effective operand address (adds the D-field, the contents of the LS register designated by the B-field, and the index, if required) for those formats that require that function.
5. Places the operands specified by RX format instructions into the applicable registers (AB, ST, and D). For the other formats, I-Fetch issues a storage request for the second operand. The second operand is placed into the registers during the execution phase.
6. Passes control to the specific execution phase by means of a 64-way branch.

The transition from the execution phase of an instruction to the I-Fetch sequence of the next instruction is achieved by an end-op cycle, the last cycle of the execution phase. The end-op cycle completes the execution phase of the instruction being processed by:

1. Setting the condition code to reflect the result of the instruction, if applicable.
2. Detecting exceptional conditions and interruptions.

The end-op cycle initiates the I-Fetch sequence for the next instruction by:

1. Decoding the format of the next instruction.
2. Initiating operand fetches as required by that format.
3. Performing a 64-way branch to establish the correct I-Fetch sequence for that format.
4. Fetching more instruction halfwords, if required.

Functional Units Used

Five registers play vital roles in the I-Fetch sequence: Q, R, E, IC, and D. The following paragraphs discuss the functions generally performed by these five registers.

Q-Register

- Buffers four instruction halfwords received from main storage.
- Provides for overlap of I-Fetch and instruction execution.
- Transfer of B- and D-fields from Q reduces instruction processing time.

The Q-register is a 64-bit (plus eight parity bits) trigger register that buffers all instructions entering the CE from main storage (Diagram 2-1, FEMDM). It is divided primarily into four halfword (16-bit) areas. This arrangement provides for the buffering of four instruction halfwords (eight bytes), thus increasing processing efficiency and reducing the length of main storage time required by the CE. The Q-register is loaded directly from the SDBO; information is transferred to the LS address register [LAL (Read) and LAR (Write)], to the R-register, and to the parallel adder.

After being loaded with a doubleword from main storage, those Q-register halfwords containing instruction op codes are sequentially transferred to R (for subsequent execution in E). When the last op-code halfword has been transferred from Q, a new doubleword is again loaded into Q from main storage. This process of continuously refilling Q with instructions is overlapped with instruction execution whenever possible.

Additional Q-register information selects the instruction fields to be sent to LAR and to the parallel adder. Such information consists of four four-bit fields (B-fields) specifying LS registers and four 12-bit fields (D-fields) containing the displacement for main storage addresses. Transferring this information directly from Q instead of via R or E provides a lookahead capability by allowing both LS and effective addresses to be available before the execution time of the associated instruction. Transferring of these 4- and 12-bit fields is performed selectively so that the information is associated with the correct instruction. A parity generator, associated with the Q-register, adjusts parity for the half-byte portion of these fields before they are transferred through the parallel adder.

Before an instruction is executed, it is tested for odd parity. The op-code halfword is tested in the E-register. The remaining halfwords, if any, are tested by the parallel adder half-sum checking circuits as the effective address is calculated.

R-Register

- Only op-code halfwords are transferred from Q to R.
- Selection of op-code halfword is determined by 1C(21,22).

The R-register is a halfword (16 bits plus two parity bits) trigger register, providing intermediate buffering of op-code halfwords between Q and E (Diagram 2-1, FEMDM). This buffering extends the total instruction buffering capability to five halfwords (five instructions in the event of all RR formats), as Q is normally refilled after the last op-code halfword has been transferred to R. The use of two separate registers (Q and R) for containing op-code halfwords also provides double buffering. This scheme allows storage requests to be generated immediately upon transferring the last op-code halfword from Q, instead of having to wait

until the instruction in E has been executed, as would be required if halfwords from Q were transferred directly to E.

Because op-code information is all that is required to initiate execution, only those halfwords in Q containing op codes are gated to R. Also, because RX, RS, SI, and SS instructions are composed of either two or three halfwords (only the first of which contains the op code), it is necessary to select the proper halfword to be transferred to R, rather than merely proceeding sequentially through the four halfwords. Selection of the halfword for transfer to R is determined by IC(21,22) as follows. Depending on the format, instructions may be 1, 2, or 3 halfwords long. The number of halfwords in an instruction is specified by the first two bits of the op code as follows:

Format	Op Code Positions 0 and 1	Instruction Length in Halfwords
RR	00	1
RX	01	2
RS and SI	10	2
SS	11	3

Because the op code of the next instruction to be executed is always in R, its format (positions 0 and 1) can be predecoded to determine the number of halfwords that compose that instruction and thus indicate which of the four Q-register halfwords contains the next sequential instruction op code. This predecoding occurs at end-op time of each instruction; the result (Q halfword number) is set into IC(21,22), which in turn selects a subsequent I-Fetch ROS word that specifies the next op-code Q-halfword to be transferred to R. The IC(21,22) values associated with each Q-register halfword are illustrated in Figure 1-13.

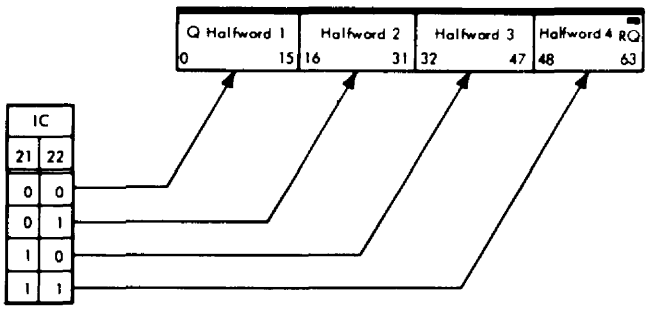


Figure 1-13. Q-Register Halfword Outgating per IC(21,22)

R(8–11) or R(12–15) is sent to LS address register to prefetch an operand for RR format instructions during end op of the preceding instruction. This transfer can be done from R rather than from Q because RR instructions are completely contained in R; eight additional paths from Q to LAL are, therefore, not needed.

E-Register

The E-register (Diagram 2-1, FEMDM) is a halfword (16 bits plus two parity bits) trigger register that contains the first halfword (op-code halfword) of the instruction being executed. Portions of the op-code halfword in E are transferred to LAL, the op-code decoder, the parallel adder, the E-register incrementer, and, if the Direct Control feature is installed, an external device. The contents of the E-register are parity-checked.

Instruction Counter

- IC is divided into two sections: IC(0–20) and IC(21–23).
- IC(0–20) addresses a doubleword from main storage.
- IC(21) specifies left or right word within accessed doubleword: IC(21) = 1, select right word; IC(21) = 0, select left word.
- IC(22) selects left or right halfword from selected word: IC(22) = 1, select right halfword; IC(22) = 0, select left halfword.
- IC(21,22) specifies Q-register halfword that contains op code of next instruction to be executed.
- During VFL operations, IC(21–23) specifies addressed byte within doubleword addressed by IC(0–20).

The Instruction Counter (IC), (Diagram 2-1, FEMDM) is a 24-bit trigger register used primarily for accessing the next sequential doubleword of instructions from main storage (excluding those specified by branch instructions, which are handled by the D-register). Source operand data is also addressed by the IC during VFL instructions.

The IC is divided into two logical sections: IC(0–20) and IC(21–23). These sections function in the following manner. The main storage area used with the CE is addressable on a byte (eight-bit) basis, each address placed into the IC referring to a particular byte. However, because the Q-register is of doubleword (64-bit) length, instructions are accessed from main storage in doubleword (eight-byte) groups. The address of the first byte of each doubleword is all that is required in accessing these doublewords from main storage, and this address is obtained from positions IC(0–20), regardless of the complete address. [Any address represented only by IC(0–20) is a multiple of 8 and lies on a doubleword integral boundary.]

Following the accessing of each doubleword from main storage, IC(0–20) is incremented by 8 (via the parallel adder) to develop the next sequential doubleword address

in main storage (eight byte addresses ahead of the doubleword previously accessed).

Once the doubleword addressed by IC(0–20) is read into the CE, the remaining portion of that complete address [IC(21–23)] selects either instruction halfwords or data bytes from the doubleword. When instructions are addressed by the IC, IC(21,22) only is used to extract the op-code halfword of the addressed instruction in the Q-register; IC(21) selects the right or left word within the doubleword, and IC(22) then selects the right or left halfword from the specified word. (In both cases, a 1 specifies the right portion and a 0 the left portion.) IC(21,22) values of 00, 01, 10, and 11 correspond respectively to the four (1–4) Q-register halfword portions. Figure 1-14 illustrates the Q-register halfword selection for a specified main storage instruction address of 468 (1D4 hex or 111010100 binary).

Note: Because instructions are restricted to even-numbered storage locations, IC(23) must always contain a 0 during instruction addressing. Detection of a 1 in IC(23) during instruction addressing produces an exceptional condition followed by a program interruption.

At end op of each instruction, the format of the instruction just transferred from Q and its location in Q [per IC(21,22)] are examined to determine the location in Q of the op-code halfword of the next sequential instruction. Both format and Q-register location must be considered to avoid transferring the remaining non-op-code halfwords of a multi-halfword instruction (RX, RS, SI, or SS) to R.

When the IC is used for addressing source operands during VFL operations, doublewords containing the addressed byte(s) are referenced by IC(0–20) in the same manner as in instruction addressing. However, the accessed doubleword is read out to AB instead of to Q. IC(21–23) then specifies the addressed byte within this doubleword to be gated to the serial adder. [The initial IC(21–23) value is

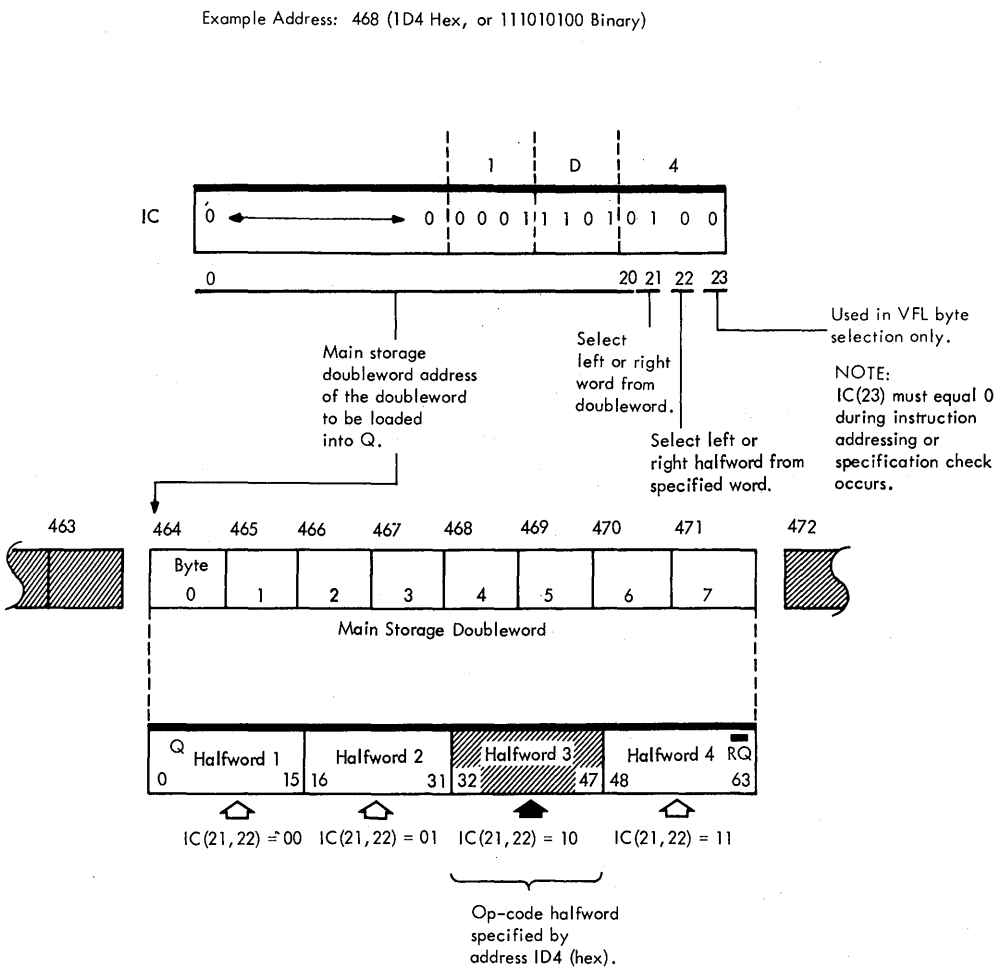


Figure 1-14. Instruction Addressing

set into the AB counter, which is incremented or decremented, as required, to perform right-to-left or left-to-right processing of the data in AB.] Figure 1-15 illustrates the byte selection as determined by IC(21–23).

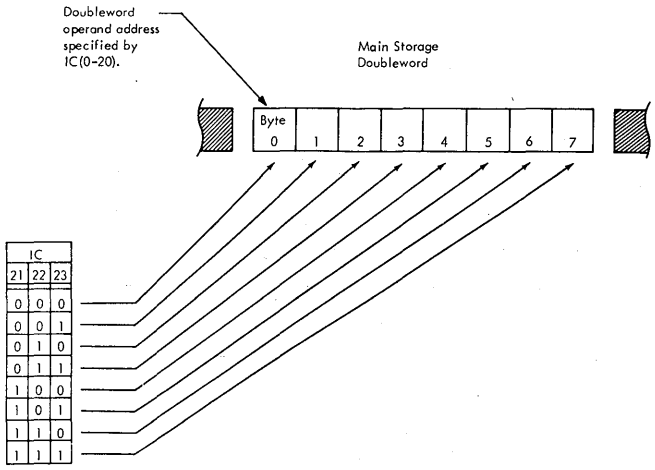


Figure 1-15. Operand Data Byte Selection per IC(21–23)

Note: During VFL operation, the instruction address contained in the IC is temporarily stored into the LS working register (LSWR).

Storage requests are generated to access the next doubleword whenever IC(21–23) indicates that all the information in the present doubleword (op-code halfword in Q-register or data bytes in AB) has been processed.

D-Register

The D-register (Diagram 2-1, FEMDM) is a 24-bit (plus three parity bits) trigger register that functions as a main storage address register during manual-control, branching, and certain arithmetic operations, and as a channel and unit address register during I/O instructions. When addressing main storage, D(0–20) references a doubleword; D(21,22) then extracts the desired instruction halfword and D(23) extracts the desired byte depending on the operation.

For RS instructions, the I-Fetch routine adds the base and displacement values, and places the result into D. Normally, this result is the effective second operand address. For shift instructions, however, this total specifies the number of bit positions to be shifted.

In the Stopped state, D contains the main storage address of the next instruction to be executed (address of instruction in R). (This address is generated and placed into D by the stop-loop microprogram that is in process whenever the CE enters the Stopped state.) The stop-loop routine subtracts 8 or 16 (decimal) from the updated IC address and places the result into D. In this case, D(0–20) indicates the doubleword address of the instructions in Q, and D(21,22) specifies the location of the op-code halfword within that doubleword.

Instruction Path

- Instructions are fetched into Q from main storage four halfwords at a time.
- R contains first halfword of instruction to be executed next.
- E contains first halfword of instruction being executed.
- IC specifies location in main storage from which next instructions will be fetched and also instruction in Q to be executed next:
  IC(0–20) addresses main storage.
  IC(21,22) indicates which op-code halfword in Q has been transferred to R and is to be executed next.
  IC(23) must be 0 when addressing instructions.

The basic path for instructions entering the CE is illustrated in Figure 1-16. The first register in the instruction path is the four-halfword instruction buffer called the Q-register. For each access, four instruction halfwords are fetched from a doubleword location in main storage (addressed by the IC) and loaded into the Q-register from the SDBO. Because instructions can vary from one to three halfwords in length, as many as four complete instructions (RR format) or as few as 1-1/3 instructions (SS format) may reside in the Q-register.

Instructions in the Q-register are sequentially selected for processing by means of IC(21,22), which indicates the first halfword (the halfword containing the op code) of the instruction to be executed next. The op-code halfword thus selected is transferred to the R-register, where format predecoding takes place during the end-op cycle. If the instruction is of the single halfword RR-format, the R-register contains the entire instruction. In the case of a two- or three-halfword instruction (RX, RS, SI, or SS format), the R-register contains only the first halfword; the balance (second or second and third halfwords) is not transferred but remains in the Q-register. For this reason, each halfword field of the Q-register is equipped with appropriate transfer paths for processing of the B and D fields of the instruction.

The format of the upcoming instruction (in R) is established by examining R(0,1). This predecoding groups the instructions into four general categories (RR, RX, RS or SI, and SS) to allow loading of the appropriate data registers with operands and operand addresses. Thus operand prefetching is initiated before execution time.

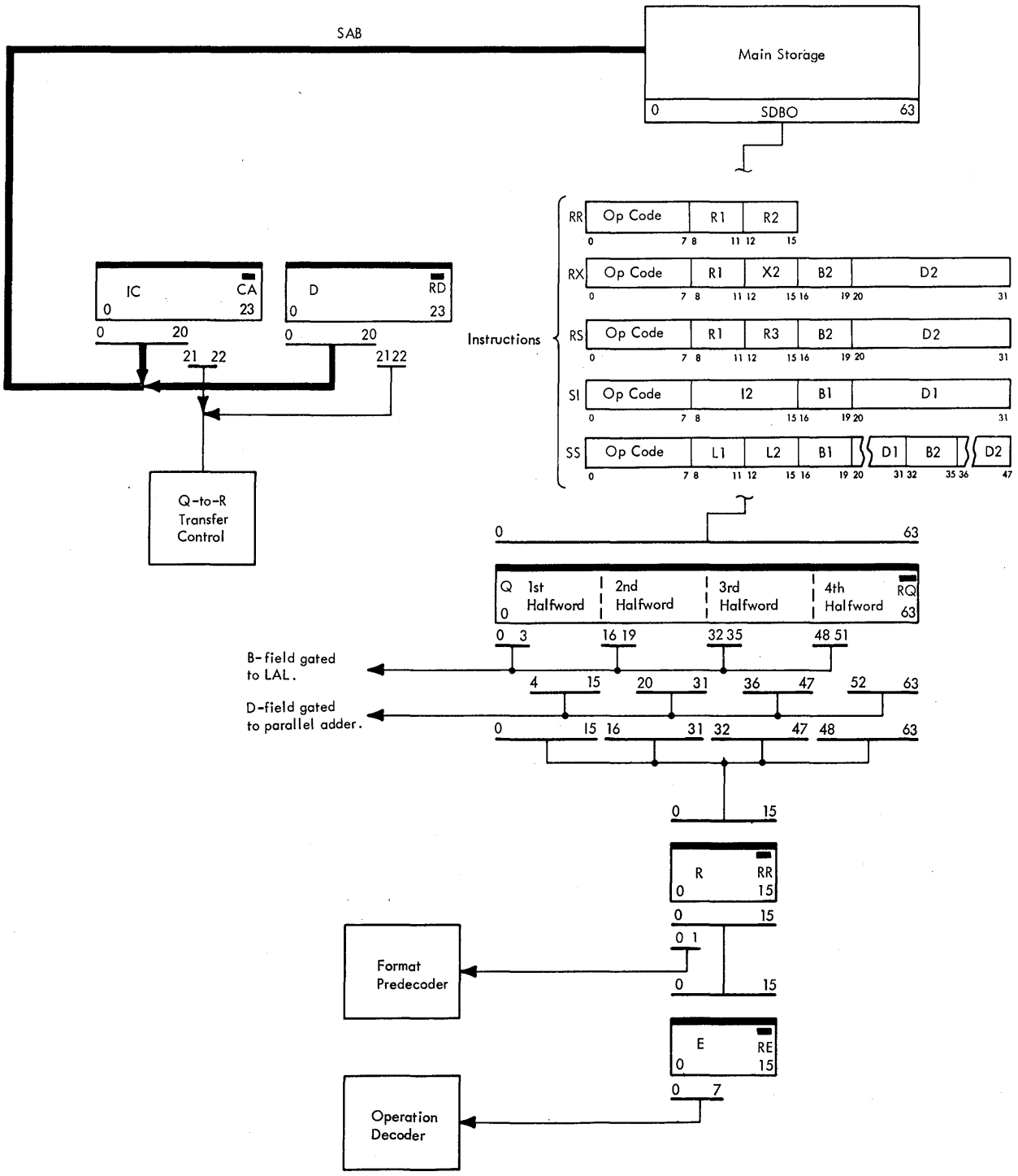


Figure 1-16. Basic Instruction Path

The 16-bit E-register (Figure 1-16) contains the op-code halfword of the instruction presently being executed. This halfword remains in the E-register until the execution phase is completed, at which time it is replaced by the op-code halfword of the next instruction. During each execution phase, the instruction op code contained in E(0–7) is decoded and the specific operations necessary to execute the instructions are performed.

The functions performed during the end-op cycle and the I-Fetch sequence are implemented while the instruction halfwords are in the Q-, R-, and E-registers. The path and the movement of the op-code halfword between the registers for the five formats are shown in Figure 1-17. To illustrate, the following paragraphs trace an RR instruction through the Q-, R-, and E-registers (Figure 1-17). Note that an RR instruction is composed of only the op-code halfword; therefore, the complete instruction fits in the R- and E- registers. For the other formats, only the op-code halfword moves through the R- and E-registers; the other halfwords are not transferred from the Q-register. For the example, assume that:

1. All instructions have the RR format.
2. The instruction being executed is No. 4, the next instruction is No. 5, the following instruction is No. 6, and so on.
3. Instruction No. 5 is the one under consideration.

During the I-Fetch sequence of No. 4, instruction No. 5 is placed into the R-register. Here, during the end-op cycle of No. 4, the format of No. 5 is established, operand refetching is initiated, No. 5 is transferred to the E-register, and the I-Fetch sequence for No. 5 is entered.

During the I-Fetch sequence for No. 5, prefetching of operands for No. 5 is completed, the execution routine for No. 5 is established, and No. 6 is transferred to the R-register. The CE enters the execution phase for No. 5.

During its end-op cycle, the condition code is set (if applicable) and any exceptional conditions and interruptions are detected. The remaining functions performed during the end-op cycle of No. 5 are devoted to initiating the I-Fetch sequence for No. 6.

Prefetching of Operands

- Operand prefetching starts before instruction execution.

- Depending on instruction format, operands are in LS or main storage:
  RR–both operands are in LS.
  RX, RS, SI–one operand is in LS, the other is in main storage.
  SS–both operands are in main storage.

- Address computation for main storage operands always starts first.

To increase the speed of instruction processing, the operands and operand addresses specified in the upcoming instruction are assembled into appropriate registers. For RR instructions, the operands are obtained directly from the LS. For instructions specifying operand addresses in main storage, address calculations take place, and the D-register prefetches an operand from main storage.

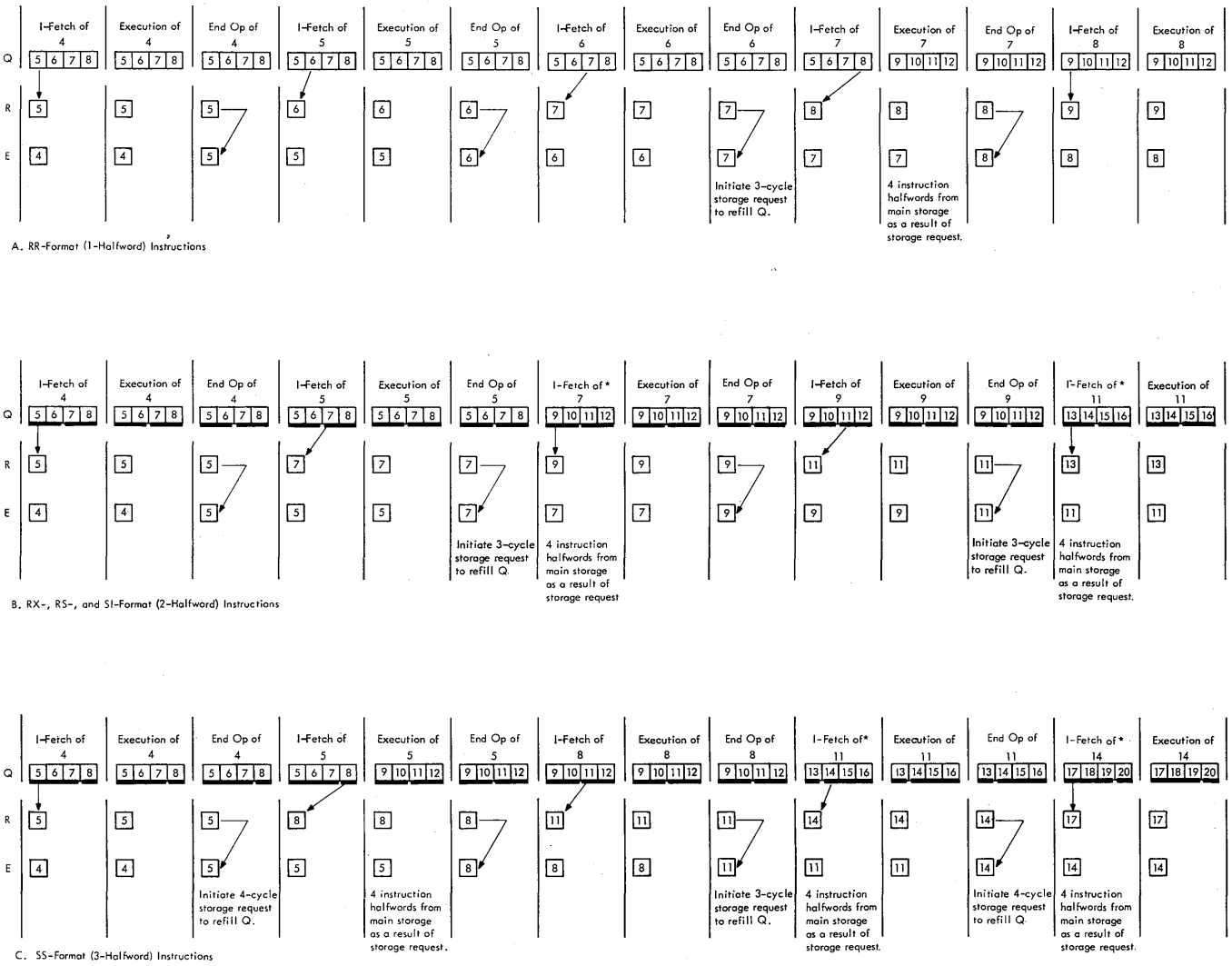
The major registers employed for operand prefetching are shown in Figure 1-18. Prefetching of operands begins when the op-code halfword of the instruction is in R and is completed after this halfword has been transferred to E. R(0,1) establishes the instruction format and, consequently, the type of operand fetch that must be performed.

For one-halfword instructions (RR format), R contains the entire instruction. The first operand is fetched by transferring R(8–11) to LAL. After the first operand is retrieved from the LS, it is usually placed into A, B, and D. The second operand is usually fetched after the instruction is transferred to E by transferring E(12–15) to LAL. When the second operand is accessed, it is normally placed into S and T.

For two-halfword instructions (RX, RS, and SI formats), the first halfword is transferred to R while the second halfword is processed directly from Q. Address calculation for the operand in main storage is performed first so that this operand may be requested as soon as possible. This calculation is accomplished by transferring the appropriate B-field from Q to LAL. If the B-field is not zero, the contents of the LS register specified by the B-field are then routed to the parallel adder, where they are added to the D-field (transferred directly from Q). The sum constitutes the operand address specified by RS and SI instructions. This address is transferred to D, from which a storage request for the operand is made.

For indexed RX instructions, that is, when the X2 field is not zero, an additional step is required to derive the operand address. Consequently, the partial sum (LS contents per B-field, plus D-field) is temporarily stored into B. The LS is then addressed by the X2 field of the instruction. At this time, the instruction op-code halfword is in E, with E(12–15) containing the X2 field. The contents of the LS register accessed by the X2 field are then summed with the contents of B in the parallel adder to obtain the operand address. This address is transferred to D, and a storage request for the operand is initiated.

The operand setup for two-halfword instructions is completed by fetching the first operand from the LS (not used by SI instructions). This action is performed by transferring E(8–11) to LAL. The first operand is usually loaded into A and B. S and T are usually loaded with the



Note: Numbers indicate halfword sequence referenced to their location in Q.

For 2- and 3-halfward instructions, lines at bottom of Q group the 4 halfwords into instructions or portions of instructions.

Movement is referenced to associated ROS control word.

* Delay transferring to R until instruction halfwords arrive from main storage.

Figure 1-17. Path Through Q-, R-, and E-Registers of Op-Code Halfword

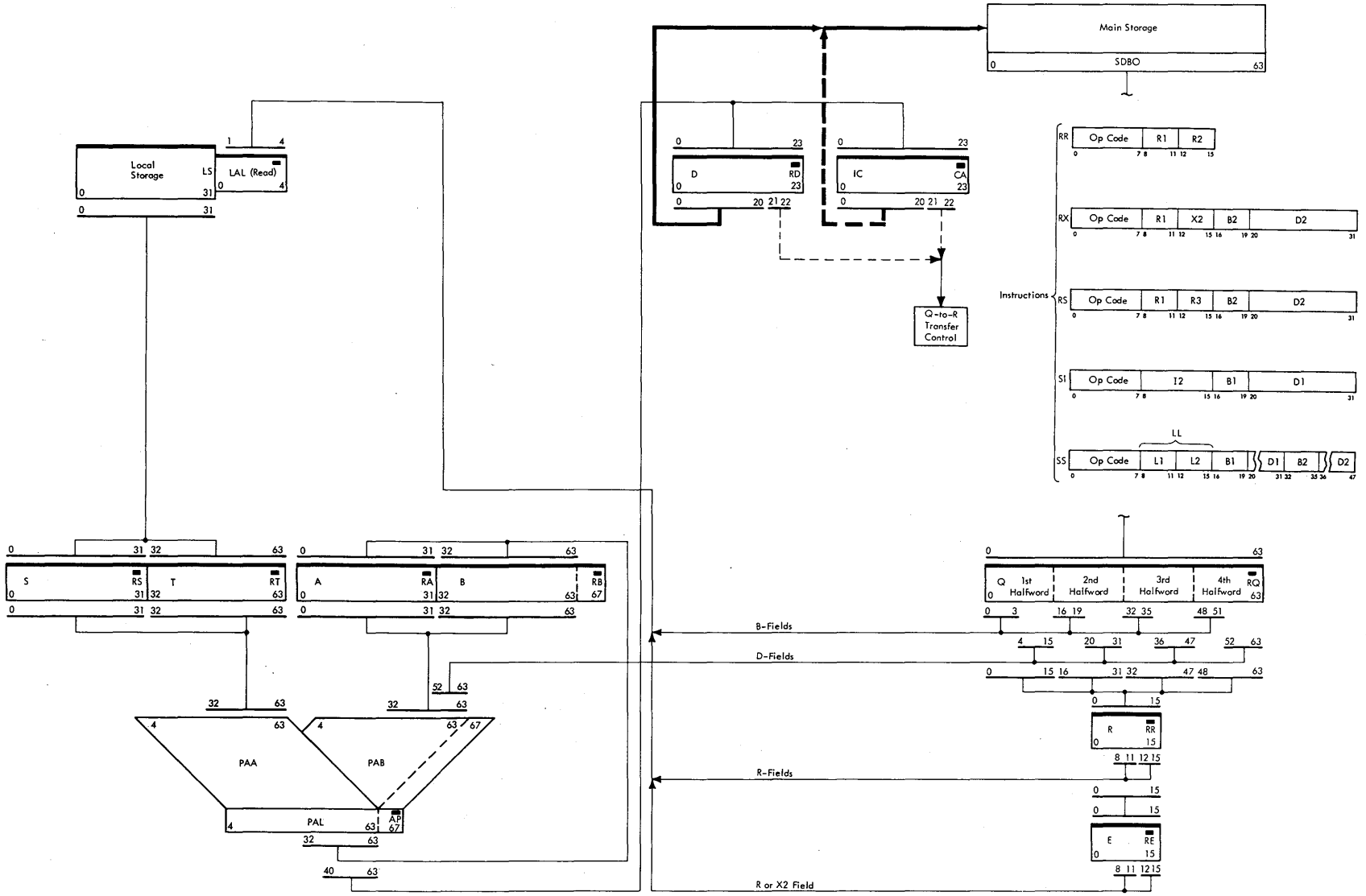


Figure 1-18. Basic Scheme for Operand Prefetching

second operand when it arrives on the SDBO during the execution phase.

For three-halfword instructions (SS format), the first halfword is transferred to R and the remaining two halfwords are processed directly from Q. The main storage addresses for the first and second operands are calculated in a manner similar to that of two-halfword instructions. The first-operand address is computed first and loaded into D, and a storage request to prefetch the operand is initiated. The second-operand address is then computed while the contents of the IC are transferred (via the parallel adder) to the LSWR. When the second-operand address is computed, the address is loaded into the IC, from which a storage request for the operand is later made. Upon execution of an SS instruction, the instruction address is restored to the IC so that it again selects the next instruction.

Obtaining New Instructions from Main Storage

- Requests for new instructions are made before CE exhausts instructions in Q.
- Usually, three- or four-cycle requests are made from IC.
- Settings of IC(21,22) and R(0,1) determine whether request is needed.
- IC is incremented by 8 after each request.
- For branch instructions, requests are made from D.
- If branch is unsuccessful, instructions accessed by request are ignored.
- If branch is successful, instructions are used and instruction address is transferred from D to IC.

Requests to refill Q with new instructions overlap most of the storage access time with processing of the remaining instructions in Q. The basic scheme used in requesting new instructions from main storage is shown in Figure 1-19.

During normal instruction sequencing, IC(21,22) is examined to establish the number of halfwords in Q that remain to be processed. These bits indicate the op-code halfword of the instruction that has been transferred to R and is to be executed next. Depending on the format of the upcoming instruction decoded from R(0,1), a request for new instructions may be initiated when 2, 3, or all 4 halfwords in Q remain to be processed. The time required to access new instructions from main storage is then used to process the remaining instruction(s) in Q. This access time is specified by the type of request generated by the CE.

Depending on its instruction status, the CE can generate a three- or a four-cycle request. When a three-cycle request is issued, three machine cycles must elapse before instructions arrive from main storage. Thus, new instructions are gated into the CE on the fourth cycle following the request. Similarly, when a four-cycle request is generated, new instructions are gated into the CE on the fifth cycle following the request. The difference in access times for a three- and four-cycle request is illustrated in Figure 1-19.

IC(0–20) normally specifies the address of new instructions to be accessed from main storage. When it is established that the CE needs instructions, a request is made, and IC(0–20) is transferred to the SAB. The IC is then incremented to address the next successive storage location from which subsequent instructions are to be fetched. (Because the SCI dictates that the address of each successive main storage location must be valid in the IC for at least two cycles, updating of the instruction address is not initiated until two cycles following the request.) Depending on the format of the upcoming instruction, incrementing of the IC is controlled by the CE hardware (for all formats except SS) or by the ROS microprogram (for SS format). In either case, the IC is incremented by transferring its contents to the parallel adder, where a 1 is added to IC(20) (equivalent to advancing the IC address by 8). The updated address is then routed back to the IC so that a new storage request may be initiated immediately upon detecting the need to refill Q.

A departure from the normal sequencing described above occurs when the instruction being fetched is a branch instruction. To anticipate branch instructions, R (which always contains the first halfword of an upcoming instruction) is connected to a branch predecoder. Execution of a branch instruction may alter the main storage address from which new instructions are to be fetched. If the branch instruction is successful, the address specified by that branch becomes the new instruction address. If, however, the branch is not successful, the address specified by that branch must be ignored. Because it is assumed that branch instructions are successful (the only exception is the Branch on Condition instructions), a request for instructions is initiated as soon as the address specified by the branch is placed into D. Thus, a request is made and the contents of D are transferred to the SAB before establishing that the branch is indeed successful. If it is later found that the branch instruction is not successful, the instructions accessed by that branch are not transferred into Q and a new storage request is generated from the IC, if necessary. Otherwise, upon establishing a successful branch, the contents of D are transferred to the parallel adder, incremented by 8, and transferred to the IC. Normal sequencing is resumed by the IC until another branch instruction is encountered.

The Branch on Condition instructions must be treated differently from other branch instructions. For this reason, a separate detection circuit is provided to anticipate this branch. Whether a Branch on Condition instruction is

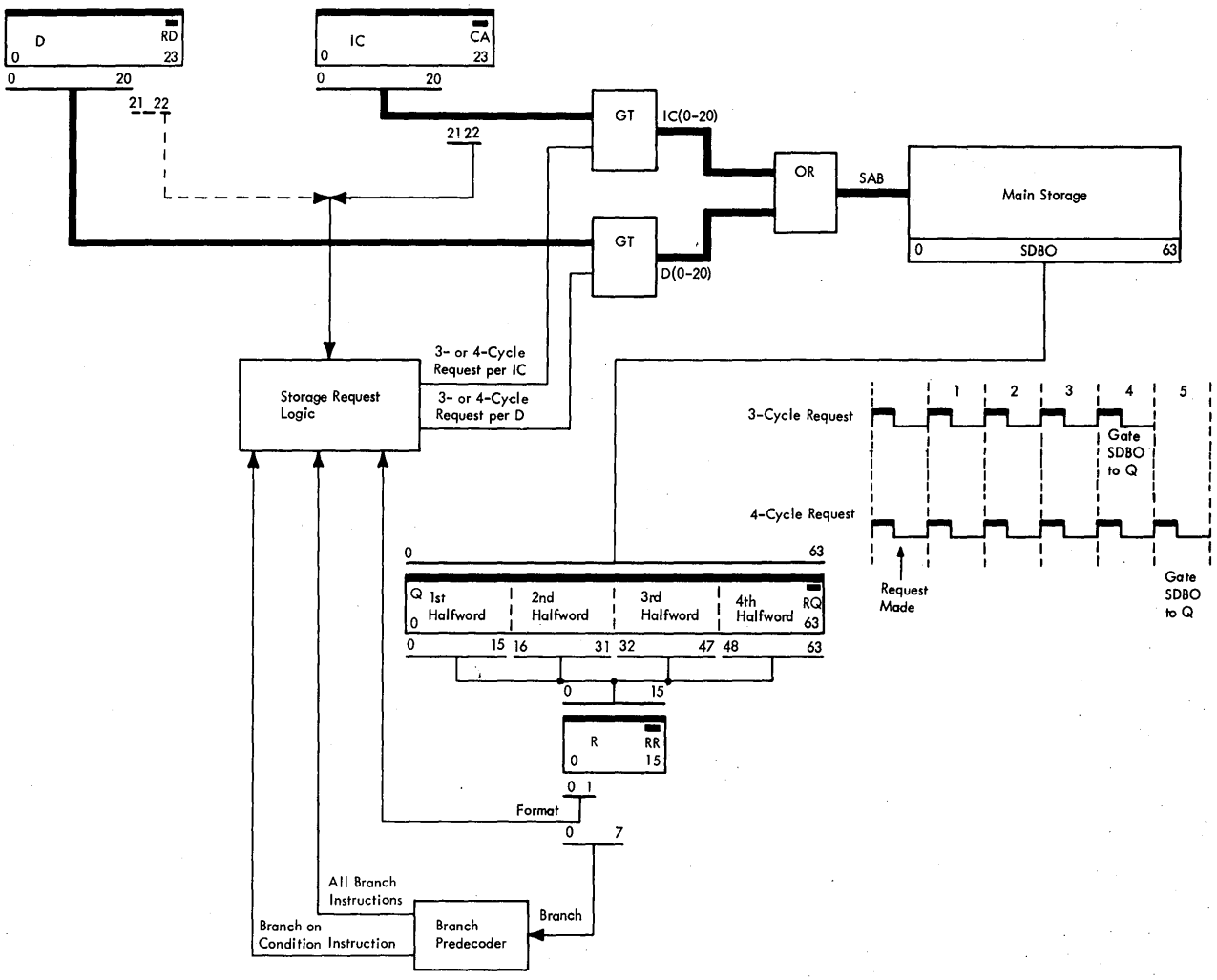


Figure 1-19. Q-Register Refill Addressing Scheme

successful depends on the condition code setting established by a previous instruction. Therefore, the outcome of this branch instruction is known at the time of the request. If the branch is unsuccessful, the request from D is inhibited and, if Q needs refilling, an IC request is generated instead.

Another instruction that falls in the branch category and requires unique treatment is the Execute instruction. This instruction designates a single instruction (subject instruction) to be inserted into the instruction sequence. Briefly, the Execute instruction initiates the following I-Fetch actions:

1. During I-Fetch of an Execute instruction (RX format), IC(21,22) is advanced to indicate the first halfword of the next instruction; that is, the instruction immediately following the Execute instruction in Q.
2. The address of the subject instruction specified by the Execute instruction is computed and placed into D. A storage request from D is then made.
3. When the four instruction halfwords accessed by this request are gated to Q, the subject instruction is selected from Q by examining D(21,22).
4. Upon executing the subject instruction, and if the subject instruction is not a successful branch, a storage request for the instructions previously contained in Q is made from the IC. This request is performed by the program store compare exceptional condition. (If the subject instruction is a successful branch, the request for new instructions is made from D and normal processing is resumed.)
5. When the instructions previously contained in Q are refetched, IC(21,22) is examined to select the instruction following the Execute instruction, and normal processing is resumed.

Interruption and Exceptional Condition Recovery

I-fetch recognizes the five classes of interruptions: external, supervisor call, program, machine check, and I/O. CE recovery from these interruptions requires additional processing time for storing the old PSW into main storage and fetching a new PSW and new instructions from main storage into the CE.

In addition, I-fetch recognizes eight exceptional conditions which require special handling by the CE. These exceptional conditions and the corresponding actions performed by the CE are shown in Table 1-6. Note that the Timer Exceptional condition actually consists of four separate conditions. Three of these (External Start, External Stop, and Receive ATR) may be considered a subset of the Timer Exceptional condition as shown in the table and explained later in the text.

When one or more interruptions or exceptional conditions occur, processing of the next instruction is delayed until all such conditions are cleared in the order of their priority.

Processing of interruptions and exceptional conditions is initiated during the first cycle of I-fetch. This cycle issues an 'EXCEP' micro-order to establish whether any interruptions or exceptional conditions have resulted from execution of the preceding instruction. If one or more interruptions or exceptional conditions did occur, the 'EXCEP' micro-order overrides the basic I-fetch actions and transfers control to an appropriate microprogram for clearing the condition that is assigned the highest priority. After processing this condition, the remaining interruptions and exceptional conditions, if any, are handled in the order of their priority. When all interruptions and exceptional conditions have been processed, the microprogram resumes the I-fetch sequence for the next instruction.

The priority of interruptions and exceptional conditions is:

1. Timer exceptional condition:
   a. External Start or Stop
   b. Receive ATR
   c. Time clock step
2. CPU Store in progress exceptional condition
3. Machine-check interruption
4. Program interruption
5. Supervisor call interruption
6. External interruption
7. I/O interruption
8. Manual control stop exceptional condition
9. Manual control wait exceptional condition
10. Manual control repeat exceptional condition
11. Program store compare exceptional condition
12. Invalid instruction address test exceptional condition
13. Q-register refill exceptional condition

Table 1-6. CE Actions in Response to Exceptional Conditions.

Exceptional Condition	CE Action
Timer:	This exceptional condition results from the 'time clock step' trigger being turned on by one of four conditions as shown below. CE actions depend on the results of ROS branching to determine which condition exists.
External Start	CE resets and performs a PSW Restart.
External Stop	CE resets and goes to the stopped state.
Receive ATR	CE performs the Receive ATR microprogram to gate data into its ATR from another CE.
Time Clock Step	CE fetches the timer value from main storage, decrements this value, and stores it back into main storage.
CPU Store in Progress	Extra cycles are added to I-fetch.
Manual Control Stop	An address is forced into ROSAR that places the CE into the stop loop.
Manual Control Wait	An address is forced into ROSAR that places the CE into the Wait state.
Manual Control Repeat	An address is forced into ROSAR that places the CE into the repeat instruction loop.
Program Store Compare	The CE obtains the address for the next instruction and refetches it from main storage.
Invalid Instruction Address Test	A program interruption is forced when the CE tries to use an erroneously addressed instruction.
Q-Register Refill	One or two extra I-fetch cycles are added, delaying execution of the next instruction.

INSTRUCTION EXECUTION

The execution phase of instruction processing manipulates the data to execute the instruction prepared by I-Fetch. The following paragraphs introduce the functional units that are used primarily in the execution phase and discuss

the nine classes of instructions: fixed-point, floating-point, decimal, logical, branching, status switching, I/O, multiple computing element, and display.

Functional Units Used

The following discussion of functional units is based on Diagram 2-1, FEMDM. For a detailed discussion, refer to Chapter 2.

AB Register

The AB register is a 64-bit (plus eight parity bits) trigger register that functions as a working register and also as a buffer for doubleword operands received from main storage.

The AB register is logically divided into two 32-bit (plus four parity bits) registers, A and B. A four-position extension, B(64–67), provides for retaining low-order significance during certain arithmetic and shifting operations.

Byte gating into and out of A and B facilitates their use with the serial adder for VFL operations. A three-position counter, the AB byte counter (ABC), controls the outgating selection of the eight bytes in the AB. AB information is also processed in the parallel adder. Both A and B outgating controls are capable of shifting data left two positions en route to the parallel adder. (Combined shifting capabilities of ST, AB, and the parallel adder thus enable any amount of shift in any direction.)

Inputs to the AB register are from storage (via SDBO) and from the parallel adder.

Outputs from the AB register are to the serial adder, the parallel adder, and the MCW register.

ST Register

The ST register is a 64-bit (plus eight parity bits) trigger register that functions as an operand buffer between main storage, LS, and the CE, and also as a working register for arithmetic and logical operations.

The ST register is logically divided into two 32-bit (plus four parity bits) registers, S and T, which serve as working registers for all operations. They also serve as a final assembly area for resultant data to be entered into either LS (T only) or main storage (S and T). Byte gating of ST inputs and outputs facilitates their use with the serial adder for VFL operations. A three-position ST byte counter (STC) and incrementer control the gating of the eight ST bytes.

ST information can also be processed in the parallel adder. T-data can be sent to the parallel adder in either true or complement form, and with a left 1 shift for certain operations. (Scan operations also utilize ST inputs from the parallel adder.) Multiply logic extracts data, in byte lengths, from S via the multiplier bus, and PSW information is sampled from ST(0–39) via the PSW bus.

Inputs to the ST register are from main storage, LS, parallel adder, serial adder, PSW register, Data switches, and the special 9020 registers.

Outputs from the ST register are to main and display storage, LS, PSW register, serial adder, parallel adder, multiply/divide logic and some special purpose registers (external, DAR, etc.).

AB and ST Byte Counters

For operations involving the serial adder, it must be possible to extract bytes from doublewords contained in AB and ST and to assemble bytes in ST for subsequent storage. These capabilities are provided by two byte counters: the ABC for controlling AB byte transfer and the STC for controlling ST byte input and output.

Mark Triggers

Eight mark triggers are contained within the CE. During store-data operations, these triggers indicate to main storage which bytes of doubleword data placed on the SDBI are to enter storage. The mark triggers thus serve to store CE data into main storage on a byte (eight-bit plus parity) basis.

An active mark bus line specifies the corresponding byte for storage entry: mark trigger 0 specifies byte 0, or SDBI(0–7); mark trigger 1 specifies byte 1, or SDBI(8–15); and so on, through mark trigger 7, which specifies byte 7 or SDBI(56–63). Any mark trigger not set causes its corresponding byte of original storage data to be regenerated into the addressed location. Complete absence of mark signals on the mark bus occurs only on a fetch operation.

F-Register

The F-register is a one-byte (eight-bit plus parity) trigger register used in certain arithmetic, logical, and data-transfer operations. Functions such as developing quotients, saving floating-point characteristics, converting routines, and processing storage-protection keys are performed in F, and, during direct-control read operations, F serves as an entry buffer for data from the sending CE.

The F-register is also utilized by the Load Identity (LDI) instruction and by diagnose operations (store PIR and define storage).

F-register parity circuits generate correct parity for all information received by the register.

G-Register

The G-register is an eight-bit trigger register used primarily for buffering a byte of data in the sending CE for transfer to another CE during direct-control write operations.

K-Register

The K-register is a 32-bit (plus 4 parity bits) register used primarily for the DE wrap function of the Diagnose instruction. DE wrap is discussed under "Maintenance Features" later in this chapter. The output of the K-register feeds the A-side of the parallel adder. During the DE wrap operation, data is set into the first or second half of the K-register via a 16-bit wrap bus from a DE. An additional input to bits 0-31 of the K-register is available from the parallel adder. This input is used during execution of the three display instructions that use the K-register: Convert and Sort Symbols (CSS), Convert Weather Lines (CVWL), and Repack Symbols (RPSB).

N-Register

The N-register is a 16-bit (plus 2 parity bits) register used in the execution of the display instruction Repack Symbols (RPSB). Inputs to the N-register are from the LM register and the serial adder. The output of the N-register feeds the B-side of the serial adder.

LM Register

The LM register is a doubleword register (64 bits plus 8 parity) used during execution of two of the display instructions: Convert Weather Lines (CVWL), and Repack Symbols (RPSB). The LM register has inputs from the Storage Data Bus Out (SDBO) and the T-register. The output feeds the XY register via the mixer. Also, any halfword of the LM register may be gated to the N-register. This output gating to the N-register is used during execution of Repack Symbols (RPSB).

Mixer

The mixer is the functional unit, consisting of logic and gating circuitry, which alters the format of data from the LM register before that data is transferred to the XY register. Gating in the mixer is under microprogram control. Any one of a number of format changes can be made during the transfer of data from LM to XY, depending on the particular micro-order that is active during the transfer. Three classes of micro-orders control mixer gating: format old (FMTO), format new (FMTN), and format weather (FMTW).

XY Register

The XY register is a doubleword register (64 bits plus 8 parity) which receives its input from the mixer. The output of the XY register can be gated to the Storage Data Bus In (SDBI) for transfer to main storage. The XY register is used during execution of two display instruction: Convert Weather Lines (CVWL) and Repack Symbols (RPSB). It receives the LM register data which has been reformatted by the mixer and buffers it preparatory to transferring it to main storage.

Serial Adder

The serial adder processes information from ST, AB, F, and N on a byte basis, and is capable of performing binary and decimal arithmetic in addition to logical AND, OR, and Exclusive-OR operations. Other miscellaneous functions performed by the serial adder logic include sign insertion and correction, digit insertion, invalid character detection, zone correction (EBCDIC and USASCII-8), zero and nonzero recognition, and parity adjustment. Parity-predict logic and carry lookahead logic are employed to improve operational speeds, with checking performed on both a half-sum and full-sum basis.

Arithmetic Functions. Highlights:

- Operates on binary or decimal data.
- Processes decimal bytes as two four-bit groups.
- Each four-bit group represents one BCD digit.
- Employs excess-6 arithmetic when processing decimal data.
- Decimal arithmetic results are produced in BCD form.
- Multiply/divide results are assembled in serial adder.

Serial adder logic is capable of performing both binary and decimal arithmetic operations. For binary operations, the adder functions as an eight-bit (plus parity) binary adder, processing bytes from either the ST register, N-register, or bits 21-24 of the ROS word (B side input) and either AB or

F (A side input). High-order carries resulting from arithmetic operations are stored in Status Trigger (STAT) H for use in processing the next two bytes of that operation.

Output from the serial adder is through the Serial Adder Latches (SALs) to the ST, F, N, or G-register.

For decimal operations, each byte of data from ST, AB, or F is treated as two individual four-bit groups, which are then processed with excess-6 arithmetic. This feature provides a programming advantage by enabling the adder to accept data in binary-coded decimal (BCD) form (packed digits) and to produce results which are also in decimal format.

Excess-6 arithmetic involves adding 6 (under ROS control) to incoming first operands. This is necessary to preserve the decimal value in a four-bit binary character. Each binary character (four bits) has a maximum decimal value of 15 (1111 binary), which is 6 more than the maximum valid decimal character of 9 (1001 binary). When the second operand is added and the total exceeds 15 (1111 binary), the carry is a decimal value carry and leaves a correct decimal value in the four binary bits. If no carry occurs after the addition, the character is an erroneous decimal value (it is too large by 6) and the excess 6 is subtracted. The decimal values of the binary character under no-carry conditions are 6–15 (actual value of 0–9 after correction).

The following examples show excess-6 addition. Note that correction occurs when there is no carry irrespective of the decimal validity of the sum.

	1 + 1	3 + 5	6 + 9
Operand 1 (SBA)	0001	0011	0110
Excess-6	0110	0110	0110
SAA	0111	1001	1100
Operand 2 (SBB)	0001	0101	1001
Sum	1000	1110	←0101
Correction	0110	0110	---
Result	0010	1000	←0101
	(2)	(8)	(15)

In the first case (1 + 1) the sum is a valid decimal character (8), but the absence of a carry indicates an erroneous result calling for subtracting the excess 6. The second case (3 + 5) is similar except that the sum is not a decimal character. In the third case (6 + 9) there is a carry into the decimal tens position and the sum (5) does not need correction.

Figure 1-20 shows the excess-6 data paths through the serial adder and illustrates the adder operation by means of the decimal example: 46 + 28 = 74. Note that +6 constants are logically added to the A-side digits that are in packed format before entry into the adder (by final-bus-A-gating logic), and that subtraction of the +6 constants (decimal-correct) is performed on each four-bit group in which a group carry (carry from high-order position of group) does not occur as a result of the arithmetic operation.

When a ROS micro-order calls for complement add, the data entering SAA is converted to 2's-complement form. This conversion is accomplished by complementing the bit configuration on the A-side entry and adding a hot carry to the input of SAL(7). For complement add decimal operation, +6 constants are not combined with SBA inputs. When complementing BCD, the excess-6 is effectively added because the resultant complement is a character based on 16 rather than 10. To illustrate, the 10's-complement of 7 is 3, but the 2's-complement of 7 (0111) is 9 (1001) or 6 more than the 10's-complement. The addition then occurs as in true +6 add, and the absence of a carry likewise forces decimal correction of the sum. A carry out of the high-order adder position [serial adder bit carry (0)] sets STAT H for use in processing the next data byte of that operation.

Figure 1-20 also shows the serial adder operation for a complement add example: 46 – 28 = 18. Note that +6 constants are not added to the A-side digits, but decimal correction is performed in the same manner as for a true add operation.

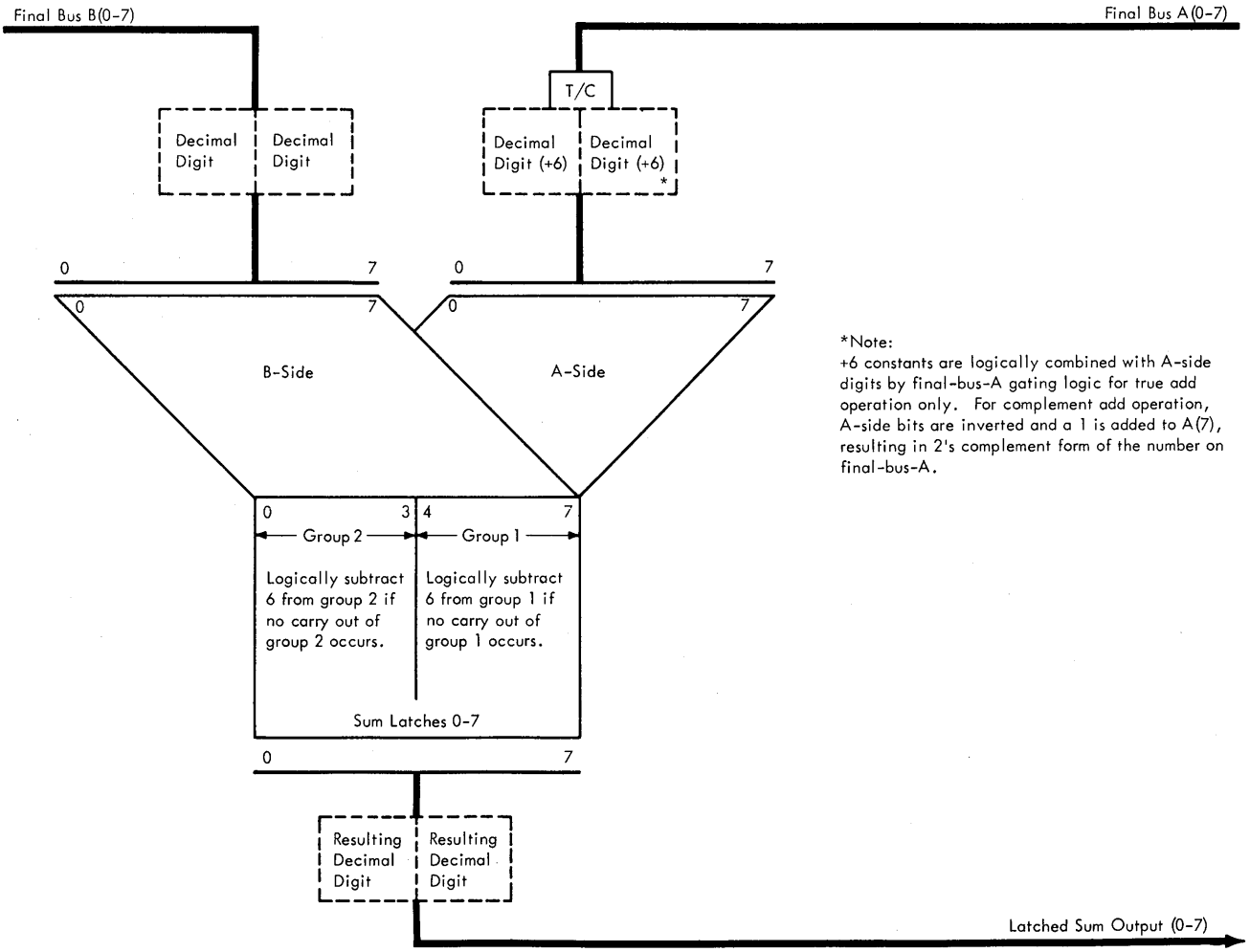
Logical Functions. The serial adder also performs logical AND, OR, and Exclusive-OR functions. To implement the logical functions, each bit position of the serial adder is, in effect, a separate unit. The logical functions are defined as follows:

1. AND. If both operand bits are 1's, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)
2. OR. If either operand bit is a 1, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)
3. Exclusive-OR. If one and only one of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)

If the conditions for the corresponding function is met, the associated serial adder latch is set.

Parallel Adder

- 60-bit (plus parity) full-binary adder.
- Inputs are from S, T, D, A, B, Q, IC, E, and F.



Decimal Arithmetic Example: 46 + 28 = 74

A-side operand byte (46_{10})	0100	0110
Convert A-side digits to excess-6 (Logically add +6 to both groups at final bus.)	0110	0110
A-side digits in excess-6 value	1010	1100
B-side operand byte (28_{10})	0010	1000

	Group 2	Group 1
A-side adder entry	1010	1100
B-side adder entry	0010	1000
Logical sum	1101	0100
Decimal correction (Logically subtract 6 from group 2.)	0110	
Decimal result (74_{10})	0111	0100

Decimal Arithmetic Example: 46 - 28 = 18

A-side operand byte (28_{10})	0010	1000
Complement	1101	0111
+1 to A(7)		1
2's complement of A-side digit	1101	1000
B-side operand (46_{10})	0100	0110

	Group 2	Group 1
A-side adder entry	1101	1000
B-side adder entry	0100	0110
Logical sum	0001	1110
Decimal correction (Logically subtract 6 from group 1)		0110
Decimal result (18_{10})	0001	1000

Figure 1-20. Decimal Format Serial-Adder Data Flow

- Inputs from T and D are 2's complemented for subtract and compare operations and address updating.
- Output data can be shifted left 4 or right 4 into the parallel adder latches; parity is adjusted accordingly.
- Adder employs carry-lookahead and parity-predict logic.
- Adder includes half-sum and full-sum error checking.

The parallel adder is a 60-bit (plus parity) full-binary arithmetic unit. In addition to arithmetic functions, the parallel adder performs certain logical operations (e.g., convert routines) and is involved in most intra-CE data transfers. Correct parity (odd) is generated with all adder output data. Immediate left 4 and right 4 shifting capabilities are available at the adder output, with parity adjusted accordingly. Error-checking facilities within the adder provide for validity checking of both incoming operands and full-sum results.

The parallel adder has true-complement gating controls on adder entries from T and D. For subtract and compare instructions, operands from T are 2's complemented and presented to the A-side of the adder. (The binary bits are inverted and a hot-carry is added to position 63.) From this point, add and subtract operations are the same. Complement entries from D are used for address compare and address update operations and for floating-point operations.

The parallel adder has full-binary capabilities (half-adder and full-sum functions), with immediate left-4/right-4 shift logic included on its output to facilitate data shifting without the need of an additional machine cycle. Figure 1-36 illustrates the logical functions of the parallel adder. Note the four-position adder extension, PAB(64–67); it serves to retain low-order significance during certain right-shift operations.

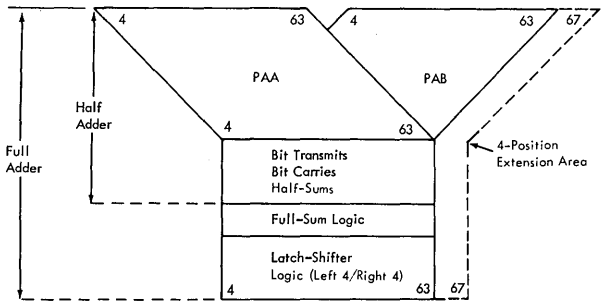


Figure 1-21. Parallel Adder Logical Functions

Half-adder functions supply information concerning incoming operands for use in both checking the validity of the operands and producing carry information for full-sum development. Full-sum logic combines half-adder outputs with carry information to produce the final or full-sum result. Parity information is also developed by logically combining half-adder and carry functions and is normally supplied on a byte basis, although certain adder areas (bits 4–8 and 64–67) require half-byte or four-bit group parity.

Carry-lookahead and parity-predict facilities are employed within the parallel adder. These features provide for the immediate development of full-sum results (and parity), without the need for additional cycles in which to incorporate carry information and generate parity. The lookahead and predict circuitry is implemented through logic in which the 60 bit positions are arranged in four-bit groups, and these groups divided into four sections. Figure 1-22 illustrates the logical grouping of the 60 bit positions.

All adder results are checked using half-sum and full-sum error-checking logic. Half-sum checking determines the validity of incoming operands by comparing the odd/even bit count with the assigned parity. Full-sum checking involves comparing the full-sum resultant bit count with the independently generated full-sum parity; an inconsistency in either causes a full-sum error.

Local Storage

A high-speed transistor storage area, local storage (LS), is located within the CE to reduce the number of main storage references required by the CE during each operation. The LS consists of 25 registers for use in storing address information, fixed-point, logical, and floating-point operands, and the IC contents (IC contents stored in LS working register, LSWR, only). Local storage data is available to the CE at 200-ns intervals. In addition to reducing the main storage reference, this access time increases operational speeds within the CE.

The 25 LS registers are grouped as follows: 16 (0–15) general-purpose registers (GPR's), 8 (16–23) floating-point registers (FPR's), and 1 (24) working register called the LSWR. Each register contains 32 data (plus 4 parity) positions, and is directly addressable by the R1, R2, R3, B1, B2, and X2 fields of the instructions, with the exception of register 24. Register 24 (LSWR) is not available to the operational program. It is reserved for use by ROS microprograms in manipulating information during execution of certain instructions. (Such applications include temporary storage for the contents of the IC when the IC is to be used as an operand address register, and temporary storage for floating-point second operands while prenormalizing the first operand.)

The eight FPR's (16–23) function as four double-length (64-bit) registers. Each double-length register consists of two single-length (32-bit) registers coupled as follows: 16 and 17, 18 and 19, 20 and 21, and 22 and 23. Only the leftmost 32 bit positions are used in short-operand floating-point instructions, with all 64 positions participating in

long-operand floating-point instructions. For either short or long operands, only the leftmost (even-numbered) registers must be addressed.

Data transferred from LS is parity checked on the LS out bus.

Local Storage Address Registers (LAL and LAR)

Two five-bit registers (LAL for Read and LAR for Write) are used in selecting the 25 individual local storage registers. Inputs to these registers are from the Q-, R-, or E-registers as well as directly from the Read Only Storage Data Register (ROSDR).

Status Triggers

The CE contains eight commonly available status triggers (STAT's) to record information that may be significant in the execution of present instruction operations. These eight triggers are designated as STAT's A–H, and retain such information as invalid digit-detection, overflow and carry conditions, and negatively signed operands. The STAT's are reset at each I-Fetch.

Certain STAT's serve multiple functions and are capable of receiving several types of information for use with different instructions. The outputs of these multiple-use triggers are distributed, via line-sense amplifiers, to the CE areas requiring this information.

Scan operations test STAT's; during scan in, the eight STAT's are set to the state of T(38) and T(54–60).

All STAT's are reset by either 'system reset' or 'I-Fetch reset' signals, with certain STAT's containing additional individual resets.

In general, all STAT's are reset and set during clock time of the basic machine cycle. However, certain clock signals that control these triggers are delayed 180 ns. This delayed operation is necessary to prevent timing problems that could arise if the detected conditions occurred too late to set a STAT with normal clock signals. All clock signals that control the STAT's are inhibited during scan-in operations by an 'FLT inhibit clock' signal.

Fixed-Point Instructions

The fixed-point instruction set performs binary arithmetic on operands serving as data, addresses, index quantities, and counts. Instructions are provided for loading, adding, subtracting, comparing, multiplying, dividing, shifting, storing, and converting from binary to decimal and from decimal to binary. Table 1-7 lists the fixed-point instructions.

For a discussion of number representation, data formats, and operand addressing, refer to Appendix D of this manual.

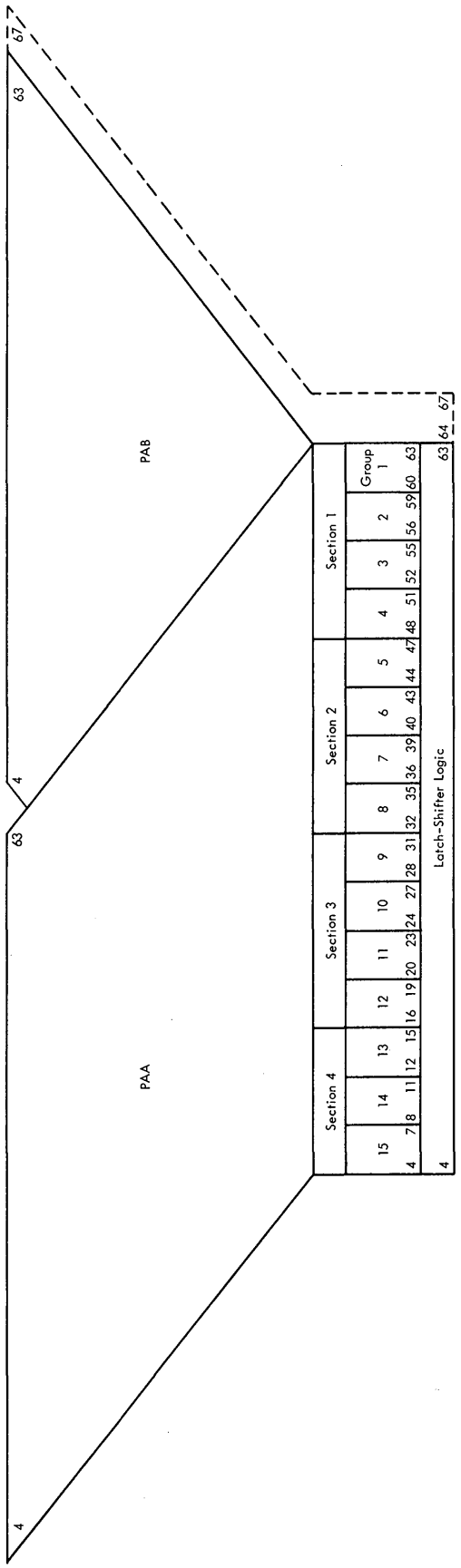


Figure 1-22. Parallel Adder Group/Section Breakdown

Table 1-7. Fixed-Point Instructions

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add	A	5A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Sum = 0 1 : Sum $<$ 0 2 : Sum $>$ 0 3 : Overflow
Add	AR	1A	RR	R1 R2	Algebraically add 2nd opr (in GPR per R2) to 1st opr (in GPR per R1) & place result into 1st opr location.	Fix-Pt Ovflo	0 : Sum = 0 1 : Sum $<$ 0 2 : Sum $>$ 0 3 : Overflow
Add Halfword	AH	4A	RX	R1 D2(X2, B2)	Algebraically add halfword 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: If 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: If 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before addition by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Sum = 0 1 : Sum $<$ 0 2 : Sum $>$ 0 3 : Overflow
Add Logical	AL	5E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word. 2. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	Prot (F) Adr Spec	0 : Sum = 0 (no carry) 1 : Sum $\neq$ 0 (no carry) 2 : Sum = 0 (carry) 3 : Sum $\neq$ 0 (carry)
Add Logical	ALR	1E	RR	R1 R2	Algebraically add 2nd opr (in GPR per R2) to 1st opr (in GPR per R1) & place result into 1st opr location. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	None	0 : Sum = 0 (no carry) 1 : Sum $\neq$ 0 (no carry) 2 : Sum = 0 (carry) 3 : Sum $\neq$ 0 (carry)
Compare	C	59	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in GPR per R1) with 2nd opr (in stg) & set CC according to result. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 $<$ Opr 2 2 : Opr 1 $>$ Opr 2
Compare	CR	19	RR	R1 R2	Algebraically compare 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & set CC according to result.	None	0 : Opr 1 = Opr 2 1 : Opr 1 $<$ Opr 2 2 : Opr 1 $>$ Opr 2
Compare Halfword	CH	49	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in GPR per R1) with halfword 2nd opr (in stg) & set CC according to result. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 $<$ Opr 2 2 : Opr 1 $>$ Opr 2

Table 1-7. Fixed-Point Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
					2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before comparison by propagating sign bit through 16 high-order bits.		
Convert to Binary	CVB	4F	RX	R1 D2(X2, B2)	Convert radix of 2nd opr (in stg) from decimal to binary & place result into 1st opr location (in GPR per R1). 1. 2nd opr is doubleword in packed format. 2. High-order word is converted first. 3. Max positive integer that can be converted is +2,147,483,647. 4. Max negative integer that can be converted is -2,147,483,648.	Prot (F) Adr Spec Data Fix-Pt Div	Unchanged
Convert to Decimal	CVD	4E	RX	R1 D2(X2, B2)	Convert radix of 1st opr (in GPR per R1) from binary to decimal & place result into 2nd opr location (in stg). 1. Result is in packed format on doubleword boundary. 2. Low-order 4 bits of field are sign. 3. If PSW(12) = 1, use USASCII-8 code for sign; if PSW(12) = 0, use EBCDIC code.	Prot (S) Adr Spec	Unchanged
Divide	D	5D	RX	R1 D2(X2, B2)	Divide 1st opr (in GPR per R1 & R1 + 1) by 2nd opr (in stg) & place result into 1st opr location (remainder in GPR per R1; quotient in GPR per R1 + 1). 1. R1 must be even adr. 2. D(21) determines which word of doubleword from stg is divisor: if 1, right word; if 0, left word. 3. Relative value of opr's must result in quotient expressible in 32-bit signed integer. 4. Sign of quotient is determined algebraically, except 0 quotient is positive. 5. Sign of remainder is same as sign of dividend, except 0 remainder is positive.	Prot (F) Adr Spec Fix-Pt Div	Unchanged
Divide	DR	1D	RR	R1 R2	Divide 1st opr (in GPR per R1 & R1 + 1) by 2nd opr (in GPR per R2) & place result into 1st opr location (remainder in GPR per R1; quotient in GPR per R1 + 1). 1. R1 must be even adr. 2. Relative value of opr's must result in quotient expressible in 32-bit signed integer. 3. Sign of quotient is determined algebraically, except 0 quotient is positive. 4. Sign of remainder is same as sign of dividend, except 0 remainder is positive.	Spec Fix-Pt Div	Unchanged

Table 1-7. Fixed-Point Instructions (cont)

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load	L	58	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in GPR per R1). 1. D(21) determines which word of doubleword from stg is to be stored: if 1, right word; if 0, left word. 2. 2nd opr is unchanged.	Prot (F) Adr Spec	Unchanged
Load	LR	18	RR	R1 R2	Load 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1). 2nd opr is unchanged.	None	Unchanged
Load & Test	LTR	12	RR	R1 R2	Load 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1) & set CC according to result. 2nd opr is unchanged.	None	0 : Result = 0 1 : Result $<$ 0 2 : Result $>$ 0
Load Complement	LCR	13	RR	R1 R2	Load 2's complement of 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1) & set CC according to result. Overflow occurs only if max negative number is 2's complemented.	Fix-Pt Ovflo	0 : Result = 0 1 : Result $<$ 0 2 : Result $>$ 0 3 : Overflow
Load Halfword	LH	48	RX	R1 D2(X2, B2)	Load halfword 2nd opr (in stg) into 1st opr location (in GPR per R1). 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before loading by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	Unchanged
Load Multiple	LM	98	RS	R1 R3 D2(B2)	Load 2nd opr (as many words as required; in stg) into GPR's, in ascending order, starting with 1st opr location (per R1) & ending with 3rd opr location (per R3). 1. 2nd opr is unchanged. 2. If R1 = R3, only 1 word is loaded. 3. If R3 $<$ R1, GPR adr's wraparound from 15 to 0. 4. D(21) determines which word of doubleword from stg is to be loaded into LS: if 1, right word; if 0, left word.	Prot (F) Adr Spec	Unchanged
Load Negative	LNR	11	RR	R1 R2	Load 2nd opr (unchanged if negative, 2's complemented if positive; in GPR per R2) into 1st opr location (in GPR per R1). If 2nd opr = 0, unchanged with plus sign.	None	0 : Result = 0 1 : Result $<$ 0

Table 1-7. Fixed-Point Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load Positive	LPR	10	RR	R1 R2	Load 2nd opr (unchanged if positive, 2's complemented if negative; in GPR per R2) into 1st opr location (in GPR per R1). Overflow occurs only if max negative number is 2's complemented.	Fix-Pt Ovflo	0 : Result = 0 2 : Result > 0 3 : Overflow
Multiply	M	5C	RX	R1 D2(X2, B2)	Multiply 1st opr (in GPR per R1 + 1) & 2nd opr (in stg) & place 64-bit result into 1st opr location (in GPR per R1 & R1 + 1). 1. R1 must be even adr. 2. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	Unchanged
Multiply	MR	1C	RR	R1 R2	Multiply 1st opr (in GPR per R1 + 1) by 2nd opr (in GPR per R2) & place 64-bit result into 1st opr location (in GPR per R1 & R1 + 1). R1 must be even adr.	Spec	Unchanged
Multiply Halfword	MH	4C	RX	R1 D2(X2, B2)	Multiply 1st opr (in GPR per R1) & halfword 2nd opr (in stg) & place low-order 32 bits of result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before multiplication by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	Unchanged
Shift Left Double	SLDA	8F	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. R1 must be even adr. 2. High-order bits of 1st opr are shifted out & lost; low-order vacated bits are made 0's. 3. If bit unlike sign bit is shifted out of bit position 1 of even register, fixed-point overflow occurs.	Spec Fix-Pt Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow
Shift Left Single	SLA	8B	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. High-order bits of 1st opr are shifted out & lost; low-order vacated bits are made 0's. 2. If bit unlike sign bit is shifted out of bit position 1 of even register, fixed-point overflow occurs.	Fix-Pt Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow

Table 1-7. Fixed-Point Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Shift Right Double	SRDA	8E	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. R1 must be even adr. 2. Low-order bits of 1st opr are shifted out & lost; high-order vacated bits are made equal to sign bit.	Spec	0 : Result = 0 1 : Result < 0 2 : Result > 0
Shift Right Single	SRA	8A	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) right number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. Low-order bits of 1st opr are shifted out & lost; high-order vacated bits are made equal to sign bit.	None	0 : Result = 0 1 : Result < 0 2 : Result > 0
Store	ST	50	RX	R1 D2(X2, B2)	Store 1st opr (in GPR per R1) into 2nd opr location (in stg). 1. PAL(61) determines into which word of doubleword in stg 1st opr is to be stored: if 1, right word; if 0, left word. 2. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store Halfword	STH	40	RX	R1 D2(X2, B2)	Store halfword 1st opr (in GPR per R1) into 2nd opr location (in stg). 1. ABC selects 16 low-order bits of 1st opr for storage; high-order bits are ignored. 2. STC [D(21–23)] positions 16 low-order bits of 1st opr into doubleword 2nd opr location. 3. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store Multiple	STM	90	RS	R1 R3 D2(B2)	Store into 2nd opr location (as many words as required; in stg) contents of GPR's, in ascending order, starting with 1st opr location (per R1) & ending with 3rd opr location (per R3). 1. GPR adr's wrap around from 15 to 0. 2. D(21) determines into which word of doubleword in stg contents of 1st GPR are to be stored: if 1, right word; if 0, left word. 3. If R1 = R3, 1 word is stored.	Prot (S) Adr Spec	Unchanged
Subtract	S	5B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Subtract	SR	1B	RR	R1 R2	Algebraically subtract 2nd opr (in GPR per R2) from 1st opr (in GPR per R1) & place result into 1st opr location.	Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow

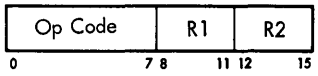
Table 1-7. Fixed-Point Instructions (cont)

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Subtract Halfword	SH	4B	RX	R1 D2(X2, B2)	Algebraically subtract halfword 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before subtraction by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Dif = 0 1 : Dif $<$ 0 2 : Dif $>$ 0 3 : Overflow
Subtract Logical	SL	5F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word. 2. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	Prot (F) Adr Spec	1 : Dif $\neq$ 0 (no carry) 2 : Dif = 0 (carry) 3 : Dif $\neq$ 0 (carry)
Subtract Logical	SLR	1F	RR	R1 R2	Algebraically subtract 2nd opr (in GPR per R2) from 1st opr (in GPR per R1) & place result into 1st opr location. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	None	1 : Dif $\neq$ 0 (no carry) 2 : Dif = 0 (carry) 3 : Dif $\neq$ (carry)

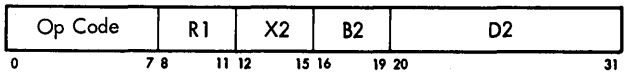
Instruction Formats

The fixed-point instruction set uses three instruction formats:

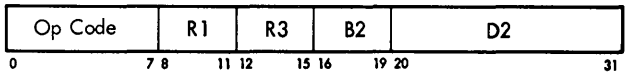
RR



RX



RS



In the RR format, R1 specifies the address of the GPR containing the first operand and R2 specifies the address of the GPR containing the second operand. Both the first and second operands may be specified by the same GPR.

In the RX format, R1 specifies the address of the GPR containing the first operand. The contents of the GPR specified by the X2 and B2 fields are added to the contents of the D2 field to form an address designating the main storage location of the second operand.

In the RS format, R1 specifies the address of the GPR containing the first operand. The contents of the GPR specified by the B2 field are added to the contents of the D2 field to form an address. This address designates the main storage location of the second operand for Load Multiple and Store Multiple instructions. In shift operations, the low-order six bits of the address specify the number of bit positions to be shifted. The R3 field specifies the address of GPR for Load Multiple and Store Multiple instruction and is ignored in the shift operations.

Data Flow

The data flow path for fixed-point operations is shown in Diagram 3-1, FEMDM. The functional units used to perform the major functions for fixed-point operations are:

1. ST. Holds the second operand and assembles data before it is sent to LS or main storage. (T is the only register that can transfer data to the LS.)
2. AB. Holds the first operand and assembles data during an operation.
3. F. Assembles product and quotient bits during multiply and divide operations; holds the binary bits to be converted during convert operations.
4. E. Controls product and quotient derivation; also contains instruction op code and number of shifts when performing shift instructions.

5. Parallel adder. Manipulates the operands to obtain the desired result. Is also the central point in the data path between ST and AB.
6. Serial adder. Calculates product and quotient bytes. Is also the central point in the data path between F and AB and between F and ST.
7. STC. Controls selection of data from and placement of data into ST, primarily during multiply, divide, and convert operations.
8. D. Addresses second operand located in main storage.

Program Interruptions

Six program interruptions can occur during execution of fixed-point instructions. Of the six, only fixed-point overflow can be masked off; the others are unconditionally taken. If the fixed-point overflow mask bit [PSW(36)] is a 0, the fixed-point overflow interruption is ignored; if a 1, it is taken.

The six interruptions and their causes are:

1. Protection. The storage key of a main storage location does not match the storage protection key in the PSW. The instruction is suppressed for a store violation, unless it is the Store Multiple instruction, which is terminated. For a fetch violation, the instruction is terminated.
2. Addressing. An address designates a location outside the available main storage capacity. The instruction is terminated except for the Store, Store Halfword, and Convert to Decimal instructions, which are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the D2 field or to the contents of the GPR's addressed by the X2 and B2 fields.
3. Specification. A data, instruction, or control word address does not specify an integral boundary for the unit of information, or the R1 field of an instruction specifies an odd register address for a pair of GPR's that contain a doubleword operand. The operation is suppressed.
4. Data. A sign or digit code of the decimal operand in the Convert to Binary instruction is incorrect. The operation is terminated.
5. Fixed-point overflow. A high-order carry occurs, or high-order significant bits are lost in load, add, subtract, or shift operations. The instruction is completed by ignoring the overflow. The interruption may be masked off by making the fixed-point overflow mask bit [PSW(36)] a 0. If the mask bit is a 1, the interruption is taken.
6. Fixed-point divide. The quotient of a division, including division by zero, exceeds the register size, or the result of the Convert to Binary instruction exceeds 31 bits. If the interruption occurs during division, the operation is suppressed. If the interruption occurs during the Convert to Binary instruction, the conversion is completed but only the low-order 32 bits of the converted data are placed into LS.

Condition Codes

The results of fixed-point load, add, subtract, compare, and shift instructions set the CC in the PSW (Table 1-7). All other fixed-point instructions leave the CC undisturbed.

For fixed-point arithmetic operations, the CC can be set to reflect three types of results:

1. For most operations, codes 0, 1, and 2 indicate the result is zero, less than zero, or greater than zero, respectively, and code 3 indicates fixed-point overflow.
2. For compare operations, codes 0, 1, and 2 indicate that the first operand is equal to, lower than, or higher than the second operand, respectively.
3. For Add Logical and Subtract Logical instructions, codes 0 and 1 indicate a zero and non-zero result, respectively, in the absence of a logical carry out of the sign position; codes 2 and 3 indicate a zero and nonzero result, respectively, with a carry out of the sign position.

Floating-Point Instructions

The floating-point instructions serve to load, add, subtract, compare, halve, multiply, divide, and store floating-point numbers. These instructions may occur in the RR format for register-to-register transfers or in the RX format for register-to-storage transfers. Eight 32-bit FPR's in LS are reserved exclusively for floating-point instructions. They are logically connected by pairs to form four 64-bit FPR's. At the end of the execution of the floating-point add, subtract, compare, and certain load instructions, a CC is set.

Operands may be either short or long. Short operands are a word long (32 bits) and long operands are a doubleword long (64 bits). Long operands provide greater precision; however, where great precision is not necessary, short operands reduce instruction execution time and the amount of storage required.

Operands and final arithmetic results are always in true form (as opposed to complement form). A 0 in the sign position indicates a positive fraction; a 1, a negative fraction. If intermediate results are in complement form, they are changed to true form before the final result is stored into the first operand location. For the add, subtract, multiply, and divide instructions, the result signs are determined algebraically.

Table 1-8 lists the floating-point instructions. For a discussion of number representation, data formats, normalization, and operand addressing, refer to Appendix D of this manual.

Table 1-8. Floating-Point Instructions

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Normalized (long)	AD	6A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (long)	ADR	2A	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2 & R2 + 1) to 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (short)	AE	7A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr; if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (short)	AER	3A	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2) to 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (long)	AW	6E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (long)	AWR	2E	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2 & R2 + 1) to 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (short)	AU	7E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

Table 1-8. Floating-Point Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Unnormalized (short)	AUR	3E	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2) to 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Compare (long)	CD	69	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in FPR per R1 & R1 + 1) with 2nd opr (in stg); CC indicates result. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Opr's remain unchanged.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (long)	CDR	29	RR	R1 R2	Algebraically compare 1st opr (in FPR per R1 & R1 + 1) with 2nd opr (in FPR per R2 & R2 + 1); CC indicates result. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Opr's remain unchanged.	Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (short)	CE	79	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in FPR per R1) with 2nd opr (in stg); CC indicates result. 1. Low-order half of FPR is ignored. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Opr's remain unchanged.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (short)	CER	39	RR	R1 R2	Algebraically compare 1st opr (in FPR per R1) with 2nd opr (in FPR per R2); CC indicates result. 1. Low-order halves of FPR's are ignored. 2. Opr's remain unchanged.	Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Divide (long)	DD	6D	RX	R1 D2(X2, B2)	Divide 1st opr (in FPR per R1 & R1 + 1) by 2nd opr (in stg) & place normalized quotient into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Opr's are prenormalized. 3. Remainder is not saved.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (long)	DDR	2D	RR	R1 R2	Divide 1st opr (in FPR per R1 & R1 + 1) by 2nd opr (in FPR per R2 & R2 + 1) & place normalized quotient into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Opr's are prenormalized. 3. Remainder is not saved.	Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged

Table 1-8. Floating-Point Instructions (cont)

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Divide (short)	DE	7D	RX	R1 D2(X2, B2)	Divide 1st opr (in FPR per R1) by 2nd opr (in stg) & place normalized quotient into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Opr's are prenormalized. 4. Remainder is not saved.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (short)	DER	3D	RR	R1 R2	Divide 1st opr (in FPR per R1) by 2nd opr (in FPR per R2) & place normalized quotient into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Opr's are prenormalized. 3. Remainder is not saved.	Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Halve (long)	HDR	24	RR	R1 R2	Divide 2nd opr (in FPR per R2 & R2 + 1) by 2 & place normalized quotient into 1st opr location (in FPR per R1 & R1 + 1). Low-order fraction of 2nd opr must be fetched from LS.	Spec Exp Unflo	Unchanged
Halve (short)	HER	34	RR	R1 R2	Divide 2nd opr (in FPR per R2) by 2 & place normalized quotient into 1st opr location (in FPR per R1). Low-order halves of FPR's are ignored & unchanged.	Spec Exp Unflo	Unchanged
Load (long)	LD	68	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in FPR per R1 & R1 + 1).	Prot (F) Adr Spec	Unchanged
Load (long)	LDR	28	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1). Low-order fraction of 2nd opr must be fetched from LS.	Spec	Unchanged
Load (short)	LE	78	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in FPR per R1). 1. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 2. Low-order half of FPR is ignored & unchanged.	Prot (F) Adr Spec	Unchanged
Load (short)	LER	38	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1). Low-order halves of FPR's are ignored & unchanged.	Spec	Unchanged

Table 1-8. Floating-Point Instructions (cont)

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load & Test (long)	LTDR	22	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1). 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0 2 : 2nd opr >0
Load & Test (short)	LTER	32	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1). 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0 2 : 2nd opr >0
Load Complement (long)	LCDR	23	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign complemented. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to original sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : Orig sign + 2 : Orig sign -
Load Complement (short)	LCER	33	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign complemented. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to original sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : Orig sign + 2 : Orig sign -
Load Negative (long)	LNDR	21	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign made minus. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0
Load Negative (short)	LNER	31	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign made minus. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0
Load Positive (long)	LPDR	20	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign made plus. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 2 : 2nd opr >0
Load Positive (short)	LPER	30	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign made plus. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 2 : 2nd opr >0

Table 1-8. Floating-Point Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Multiply (long)	MD	6C	RX	R1 D2(X2, B2)	Multiply 1st opr (in FPR per R1 & R1 + 1) & 2nd opr (in stg) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Prot (F) Adr Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (long)	MDR	2C	RR	R1 R2	Multiply 1st opr (in FPR per R1 & R1 + 1) & 2nd opr (in FPR per R2 & R2 + 1) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (short)	ME	7C	RX	R1 D2(X2, B2)	Multiply 1st opr (in FPR per R1) & 2nd opr (in stg) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). 1. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 2. Opr's are prenormalized.	Prot (F) Adr Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (short)	MER	3C	RR	R1 R2	Multiply 1st opr (in FPR per R1) & 2nd opr (in FPR per R2) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Spec Exp Ovflo Exp Unflo	Unchanged
Store (long)	STD	60	RX	R1 D2(X2, B2)	Store 1st opr (in FPR per R1 & R1 + 1) into 2nd opr location (in stg). 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store (short)	STE	70	RX	R1 D2(X2, B2)	Store 1st opr (in FPR per R1) into 2nd opr location (in stg). 1. PAL(61) determines into which half of doubleword in stg 1st opr is to be stored: if 1, right half; if 0, left half. 2. Low-order half of FPR is ignored. 3. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Subtract Normalized (long)	SD	6B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (long)	SDR	2B	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2 & R2 + 1) from 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

Table 1-8. Floating-Point Instructions (cont)

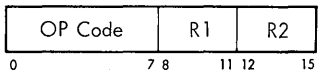
Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Subtract Normalized (short)	SE	7B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (short)	SER	3B	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2) from 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (long)	SW	6F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (long)	SWR	2F	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2 & R2 + 1) from 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (short)	SU	7F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (short)	SUR	3F	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2) from 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

Instruction Formats

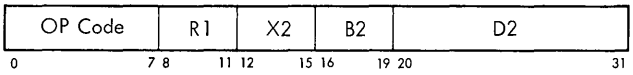
- Floating-point instructions use RR and RX formats.
- Programmer must specify even FPR of even/odd pair (0, 2, 4, or 6).
- Main storage address of second operand must designate word boundaries (bits 22 and 23 = 00) for short operands and doubleword boundaries (bits 21, 22, and 23 = 000) for long operands.

Floating-point instructions occur in the RR and RX formats:

RR Format



RX Format



In these formats, R1 is the address of an FPR that contains the first operand. The second operand location is defined differently for the two formats.

In the RR format, R2 is the address of an FPR containing the second operand. The same FPR may be specified for the first and second operands.

The R1 and R2 fields must specify 0, 2, 4, or 6, or a specification program interruption occurs. The specification check is made by testing E(8) and E(11) for zero; for RR instructions, E(12) and E(15) are tested for zero. If E(11) or E(15) does not equal zero, an odd address has been specified. If E(8) or E(12) does not equal zero, the specified FPR address is greater than 7. Thus, if any of the tested E bits equals 1, a specification program interruption is taken.

In the RX format, the contents of the GPR's specified by X2 and B2 are added to the contents of the D2 field to form an effective address designating the main storage location of the second operand. A zero in an X2 or B2 field indicates that no index or base component is to be used. The main storage address should designate word boundaries for short operands (bits 22 and 23 = 00) and doubleword boundaries (bits 21, 22, and 23 = 000) for long operands. Otherwise, a specification program interruption occurs.

The results replace the first operand except for store operations, where result replaces the second operand. Except for the storing of the final result, the contents of all LS registers and main storage locations participating in operand addressing or operation execution remain unchanged.

Data Flow

- Eight 32-bit LS registers are reserved for floating-point instructions.
- Micro-orders control low-order fraction fetch.
- LS FPR address specified must be even (0, 2, 4, or 6).
- Sign-handling is achieved via serial adder or STAT's.
- Characteristic-handling is performed via serial adder.
- Fraction-handling is performed via parallel adder.

Eight 32-bit LS registers (addresses 16–23) are reserved for floating-point instruction operands and results (Diagram 3-2, FEMDM). An even/odd pair of these registers functions as a double-length (64-bit) register with an assigned address of 0, 2, 4, or 6. A 0 in the R1 or R2 field of a floating-point instruction specifies LS locations 16 and 17; a 2 specifies locations 18 and 19; a 4 specifies locations 20 and 21; a 6 specifies locations 22 and 23.

In instructions other than floating-point, addressing is limited to 16 GPR's because the R1 and R2 fields contain four bits each. The LS address register (LAL), however, contains five bits; LAL(0) is used to address the FPR's. Because floating-point instructions must specify an LS address of 0, 2, 4, or 6 in the R1 and R2 (RR only) fields, and use only the FPR's for operands, a 1 is forced into LAL(0) when accessing LS during execution of a floating-point instruction. (Note that, for RX format instructions, the base and index register fields specify GPR's.) For example, if address 0 is specified by the R1 or R2 field, LS accesses LS register 16 (LAL = 10000). Short operand instructions fetch only 32 bits (single word) from the specified FPR. Because ingating and outgating of LS are limited to 32 bits each, long floating-point operands must be divided into two 32-bit words stored in an even/odd pair of FPR's. Under micro-order control, a 1 is forced into the low-order bit position of LAL [LAL(4)] to fetch or store the low-order 32 bits of a long operand from R1 plus 1 or R2 plus 1. For example, the 'RF*E3Ω1' micro-order specifies the FPR addressed by E(12–15) + 1. The R1 + 1 and R2 + 1 registers are the odd-numbered addresses of FPR's.

At the beginning of the execution phase of floating-point instructions, a specification test establishes that:

1. An even register is specified in the R1 and R2 (RR format only) fields.
2. A register address greater than 6 is not specified in the R1 and R2 (RR format only) fields.
3. The effective main storage address is on a doubleword boundary for long operands and on a word boundary for short operands.

Data flow may be divided into two paths: the fraction path and the sign and characteristic path. The fractions are transferred, added, or shifted via the parallel adder. The operands are located in DT, ST, and AB. For floating-point instructions, the parallel adder shifts operands right or left four bit positions under micro-order control. For floating-point, the contents of PAL can be gated to T, D, A, and B.

The sign and characteristic path is from ST or AB to F, via the serial adder. The byte gated to the inputs of the serial adder depends upon the STC and the ABC values. For floating-point, the STC is normally set to 4 to specify the first byte of T, and the ABC is set to 0 to specify the first byte of A. The data from F is gated to the serial adder. From the serial adder, the data is transferred to ST per the STC. For floating-point operations, the serial adder adds 1 to, or subtracts 1 or 64 from, the characteristic at the inputs of the serial adder under micro-order control.

In floating-point instructions, the signs are saved in STAT's under micro-order control. When the 'SAVE SIGNS' micro-order is executed, bit 0 of the ST byte selected by the STC is gated in true or complement form, depending upon the instruction, to STAT C. Because the STC is set to 0 or 4 before issuing the 'SAVE SIGNS' micro-order, the contents of either S(0) or T(32), whichever contains the sign of the operand, is saved in STAT C via the serial adder. At the same time, the sign of the operand in AB is sent to STAT F. If the instruction is a multiply or divide and SAL(0) = 1 (indicating a carry resulted from the characteristic addition or subtraction), STAT D is set.

When the results of the execution of a floating-point instruction are to be stored, the STAT's are decoded, under micro-order control, to determine the sign of the result. If the sign is minus, the 'INSERT SIGN' micro-order forces a 1 to bit 0 of the FPR (on the LS bus in) addressed by R1 and inhibits gating of T(32) to LS. The result sign is minus under the following conditions:

1. Multiply or divide and signs (STAT's C and F) are unlike.
2. Load complement and STAT C equals 0.
3. Halve, load, or load and test, and STAT C equals 1.
4. Add, subtract, or compare and sign of the larger operand is minus.
5. Load negative.

Program Interruptions

Seven program interruptions can occur during execution of floating-point instructions. Of the seven, "exponent underflow" and "significance" can be masked off; the others are unconditionally taken. If the associated mask bit [PSW(38) and PSW(39), respectively] is a 0, the interruption is ignored; if a 1, it is taken.

The seven interruptions and their causes are:

1. Protection. The storage key does not match the protection key in the PSW for all RX instructions. When an instruction causes a fetch-protection violation, instruction execution is terminated, the program execution is altered by a program interruption, and a protection program interruption is indicated in the old PSW. When an instruction causes a store-protection violation, the operation is suppressed.
2. Addressing. An address designates a location outside the available storage for the installation. The operation is terminated.
3. Specification. A short operand is not located on a word boundary, a long operand is not located on a double-word boundary, or an FPR address other than 0, 2, 4, or 6 is specified. The instruction is suppressed. The address restrictions do not apply to the components (contents of the D2 field and the contents of the LS registers specified by X2 and B2) from which an address is generated.
4. Exponent overflow. The result exponent (characteristic) of an addition, subtraction, multiplication, or division overflows, and the result fraction is not zero. The operation is completed by making the characteristic 128 smaller than the true result; the sign and fraction remain unchanged.
5. Exponent underflow. The result of an addition, subtraction, multiplication, or division underflows, and the result fraction is not zero. A program interruption occurs if the exponent-underflow mask [PSW(38)] is a 1. The operation is completed by replacing the result with a true zero, if the mask is off. If the mask is on, the characteristic is made 128 larger than the true result and the sign and fraction remain unchanged.
6. Significance. The result fraction of an addition or subtraction is zero. A program interruption occurs if the significance mask [PSW(39)] is a 1. The mask bit also affects the result of the operation. When the significance mask bit is a 0, the operation is completed by replacing the result with a true zero. When the significance mask bit is 1, the operation is completed without further change to the characteristic of the result. In either case, the CC is set to 0.
7. Floating-point divide. Division by a number with a zero fraction is attempted. The division is suppressed, but the CC and the data in storage remain unchanged.

Condition Codes

The results of floating-point add, subtract, compare, and certain load operations set the CC (Table 1-8). Multiplication, division, and storing leave the CC unchanged.

The CC can be set to reflect two types of results for floating-point arithmetic. For most operations, CC's of 0, 1, and 2 respectively indicate that the result register contains zero, less than zero, and more than zero. A zero result is indicated whenever the result fraction is zero, including a forced zero. A CC of 3 is never set by floating-point instructions.

For compare instructions, CC's of 0, 1, and 2 respectively indicate that the first operand is equal to, lower than, and higher than the second operand.

Decimal Instructions

The decimal instructions provide for addition, subtraction, comparison, multiplication, division, and format conversion of variable-field length (VFL) operands. The VFL data, which may range from 1 to 16 bytes in length, resides in main storage only. All decimal instructions are therefore in the SS format to provide for storage-to-storage operations. In general, most decimal instructions require fetching the operands from main storage, performing the operations specified by the instruction op code, and storing the results in main storage. A list of the decimal instructions is contained in Table 1-9.

For a discussion of number representation, data formats, and operand addressing, refer to Appendix D in this manual.

Data Handling

- Decimal arithmetic is performed by either true add or complement add sequence, using excess-6 arithmetic.
- True add sequence adds 6 to each digit gated to A-side of serial adder.
- Complement add sequence gates 2's complement of each digit to A-side of serial adder.
- Inputs to B-side of serial adder are unchanged.
- Each digit which did not cause a carry at output of serial adder is reduced by 6 (decimal corrected).
- If no carry from high-order digit, result is in complement form and must be recomplemented.

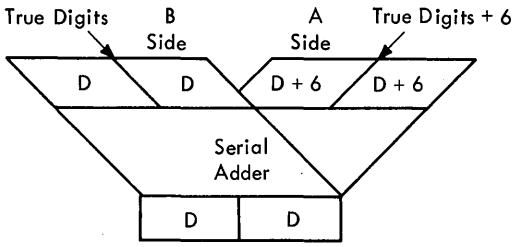
Decimal arithmetic operations are performed in the serial adder on a byte basis. A true add or a complement add sequence is used depending upon the instruction and the operand signs. Because decimal digits are in BCD format, excess-6 arithmetic is used.

As stated previously, the decimal digits are represented by a binary code. Each digit consists of a four-bit field, bit combinations 0000–1001 corresponding to decimal digits 0–9. This system of decimal notation allows relatively simple binary techniques to be applied when operating with decimal data, and also facilitates direct reading of decimal results. However, two problems are encountered. One problem is that the four-bit field used to represent decimal digits has 16 possible codes, of which 6 (binary combinations for 10 through 15 inclusive) are invalid as decimal digits. Thus means must be provided to correct invalid results when they occur in an arithmetic operation. For example, the addition of decimal digits 0110 (six) and 0101 (five) must yield a decimal result of 0001 0001 (eleven). If, however, a pure binary addition is carried out, it will yield an unacceptable result:

0110	(decimal, or binary 6)
0101	(decimal, or binary 5)
1011	(invalid as decimal, but 11 in binary)

The second problem is in the generation of a decimal carry. When the sum of two decimal digits exceeds 9, a carry must be sent to the next high-order digit. However, a pure binary addition does not yield a carry unless the sum of the digits exceeds 1111 (15), which has the effect of a hex carry; i.e., carrying the order of 16 rather than 10.

Both of the above problems are solved by the excess-6 arithmetic scheme and the decimal correction functions of the serial adder. In the excess-6 scheme, often referred to as true +6 arithmetic, a 6 is added to each digit as it is gated to the A-side of the adder, one byte (two digits) at a time from the second operand; the digits gated to the adder B-side, one byte at a time from the first operand, are not affected:



If the sum of the two digits to be added is 10 or greater, the true +6 scheme automatically eliminates the unwanted binary configuration and also supplies a decimal carry in

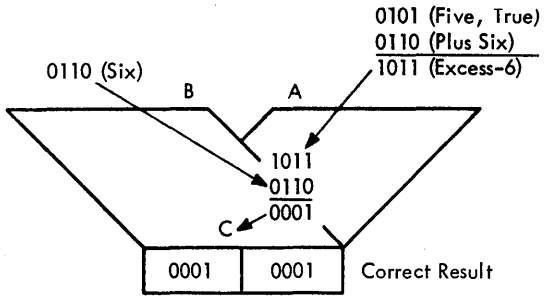
Table 1-9. Decimal Instructions

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Decimal	AP	FA	SS	D1(L1, B1) D2(L2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in stg) & place result into 1st opr location. 1. Opr's & result are in packed format. 2. Opr fields may overlap if low-order bytes coincide. 3. Right to left, byte by byte. 4. Shorter opr is extended with high-order 0's. 5. 1st opr field must be large enough to contain all 2nd opr significant digits.	Prot (S,F) Adr Data Dec Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Compare Decimal	CP	F9	SS	D1(L1, B1) D2(L2,B2)	Algebraically compare 1st opr (in stg) with 2nd opr (in stg) & set CC according to result. 1. Opr's are in packed format. 2. Shorter opr is extended with high-order 0's. 3. Opr fields may overlap if low-order bytes coincide. 4. Right to left, byte by byte. 5. Result is not stored & opr fields are unchanged.	Prot (F) Adr Data	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Divide Decimal	DP	FD	SS	D1(L1, B1) D2(L2, B2)	Divide 1st opr (in stg) by 2nd opr (in stg) & place result into 1st opr location (quotient is leftmost in 1st opr location; remainder, rightmost). 1. Opr's are in packed format. 2. Dividend must contain at least 1 high-order 0. 3. Max dividend field = 16 bytes (31 digits & sign); L1 = 15. 4. Max divisor field = 8 bytes (15 digits & sign); L2 = 7. 5. Divisor field must be < dividend field (L2 < L1). 6. Max quotient field = 15 bytes. 7. Quotient field = dividend field minus remainder (divisor) field (L1 minus L2). 8. Remainder field = divisor field. 9. Opr fields may overlap if low-order bytes coincide. 10. Sign of quotient is determined algebraically, except 0 result is positive. 11. Sign of remainder is same as dividend sign.	Prot (S,F) Adr Spec Data Dec Div	Unchanged
Move with Offset	MVO	F1	SS	D1(L1, B1) D2(L2, B2)	Store 2nd opr (in stg) to left of and adjacent to low-order 4 bits of 1st opr (in stg). 1. Opr's are in packed or unpacked format. 2. If 2nd opr is shorter than 1st opr, fill 1st opr field with high-order 0's. 3. If 2nd opr is longer than 1st opr, ignore excess 2nd opr high-order digits. 4. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged

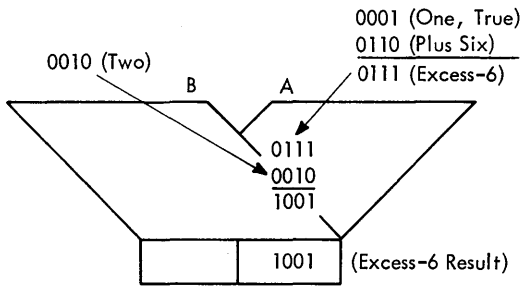
Table 1-9. Decimal Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Multiply Decimal	MP	FC	SS	D1(L1, B1) D2(L2, B2)	Multiply 1st opr (in stg) by 2nd opr (in stg) & place result into 1st opr location. 1. Opr's are in packed format. 2. Product must contain at least 1 high-order 0. 3. Max multiplicand field = 16 bytes (31 digits & sign); L1 = 15. 4. Max multiplier field = 8 bytes (15 digits & sign); L2 = 7. 5. Multiplier field must be < multiplicand field (L2 < L1); max value of L2 = 7. 6. Multiplicand field initially contains high-order 0-field equal in length to multiplier field. 7. Max product field = 16 bytes (31 digits & sign). 8. Sign of product is determined algebraically, except 0 result is positive.	Prot (S,F) Adr Spec Data	Unchanged
Pack	PACK	F2	SS	D1(L1, B1) D2(L2, B2)	Convert format of 2nd opr (in stg) from zoned to packed & place result into 1st opr location (in stg). 1. 2nd opr is in zoned format. 2. No restriction on overlapping fields. 3. Extend 2nd opr with high-order 0's, if necessary. 4. If 1st opr field is too short to contain all significant digits of 2nd opr field, ignore excess 2nd opr high-order digits. 5. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Subtract Decimal	SP	FB	SS	D1(L1, B1) D2(L2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in stg) & place result into 1st opr location. 1. Opr's & result are in packed format. 2. Opr fields may overlap if low-order bytes coincide. 3. 1st opr field must be large enough to contain all 2nd opr significant digits. 4. Shorter opr is extended with high-order 0's. 5. Right to left, byte by byte.	Prot (S,F) Adr Data Dec Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Unpack	UNPK	F3	SS	D1(L1, B1) D2(L2, B2)	Convert format of 2nd opr (in stg) from packed to zoned & place result into 1st opr location (in stg). 1. 2nd opr is in packed format. 2. No restriction on overlapping fields. 3. Extend 2nd opr with high-order 0's, if necessary. 4. If 1st opr field is too short to contain all significant digits of 2nd opr field, ignore excess 2nd opr high-order digits. 5. If PSW(12) = 1, use USASCII-8 code for zones; if PSW(12) = 0, use EBCDIC. 6. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Zero & Add	ZAP	F8	SS	D1(L1, B1) D2(L2, B2)	Place 2nd opr (in stg) into 1st opr location (in stg). 1. 2nd opr is in packed format. 2. Opr fields may overlap if low-order byte of 1st opr coincides with or is to the right of low-order byte of 2nd opr. 3. 1st opr field must be large enough to contain all 2nd opr significant digits.	Prot (S,F) Adr Data Dec Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow

terms of a hex carry. In true +6 arithmetic, the previous add example of digits 5 and 6 is executed as follows:

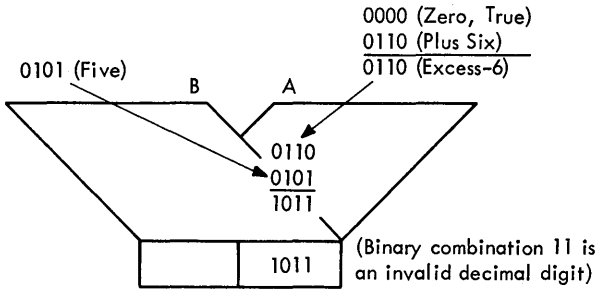


Addition of a 6 in all cases, however, may create an erroneous and sometimes invalid result. This occurs if the sum of the two digits to be added is less than 10. For example, consider the addition of decimal digits 1 and 2:



In the above case, the result (9) is clearly in excess-6 form; the digit 6 must be subtracted from the result to obtain the correct answer.

A further example illustrates how an excess-6 digit may generate an invalid result. Consider the addition of decimal digits 0 and 5:

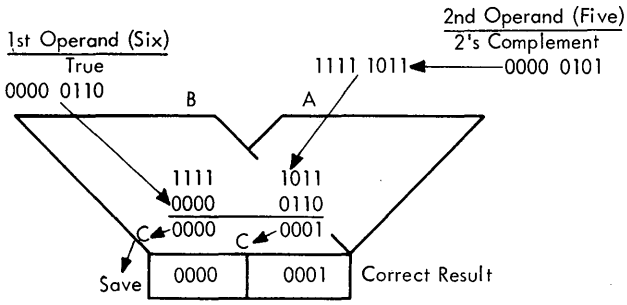


Note that both the erroneous and the invalid results are characterized by a no-carry to the next high-order digit. This condition holds true in all cases when incorrect data is generated, and is utilized by the decimal correction logic of the adder. When a no-carry condition is detected, this logic automatically deducts 6 from the result, thus supplying the correct digit to the adder output.

The decimal correct function of the adder is also used during complement add operations. The binary codes of the decimal digits at the adder A-side are gated in 2's complement form; excess 6's are not supplied. The digits at the B-side of the adder are gated in true form. The result of a complement add operation may be in true or complement form.

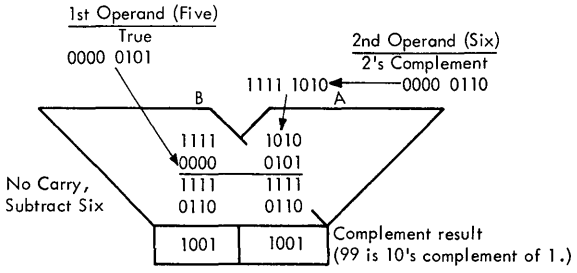
For clarity, the previous examples have shown operations that use only one digit. However, the serial adder normally handles one byte (two digits) at a time. To demonstrate the operation of the serial adder during decimal operations, the following examples deal with a byte of data.

If the first operand is larger than the second, the result is in true form. Consider complement addition of decimal digit 5 to 6; that is 6 minus 5:



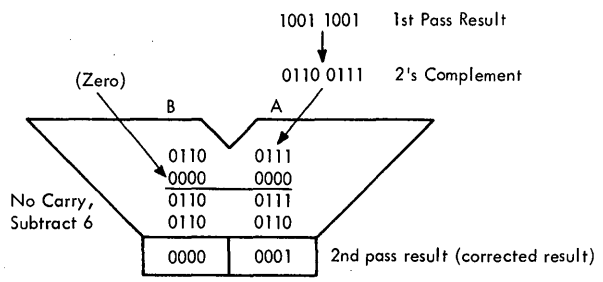
A true result during a complement add operation is always characterized by a carry from the last high-order digit. As in the case of the true add operation, a carry to the next digit indicates that no decimal correction of that digit is necessary.

If the first operand is smaller than the second, the result is in complement form. Because decimal data is always stored in true form, the result must be recomplemented. Consider complement addition of the decimal digit 6 to 5; that is, 5 minus 6:



Note that the decimal correction feature of the adder always subtracts 6 from each digit position which does not produce a carry. In a complement add operation, a no-carry condition from the last high-order digit also indicates that

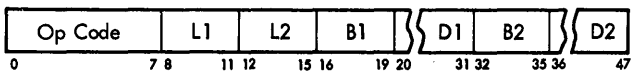
the result is in complement form and must be recomplemented. This requires a second pass through the adder:



Instruction Format

- Instructions specify two addresses.
- B1 (contents) + D1 + L1 specifies rightmost byte of 1st operand.
- B2 (contents) + D2 + L2 specifies rightmost byte of 2nd operand.
- Results are stored in true form at first operand location.

All decimal instructions use the SS format:



An SS instruction operates on two operands in main storage and stores the result into the same location from which the first operand was obtained. Therefore, the address of the first operand is also the destination address; the address of the second operand is commonly referred to as the source address.

The contents of the GPR specified by the B1 field are added to the D1 field to form an address. This address specifies the leftmost byte of the first operand. The number of operand bytes to the right of this byte is specified by the L1 field of the instruction. The L1 field may specify up to 16 bytes. Similarly, the address of the second operand is specified by the B2, D2, and L2 fields of the instruction. A zero in the B1 or B2 fields indicates the absence of the corresponding address component.

Normally, decimal operands are processed from right to left. Thus the address for the initial operand fetch is:

LS register per B-field + D-field + L-field.

Operands are fetched from main storage one doubleword, or eight bytes, at a time. Because the L-field may specify up to 16 bytes, several operand fetches may be required to completely access the operand. After each fetch, the operand address is decremented by 8 to access the next high-order eight bytes of the operand.

The results of decimal operations are placed into the first operand field and must be in true form. The result is never stored outside the first operand field specified by the instruction. If the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand. Such extension does not modify the second operand in main storage, where it remains unchanged.

Data Flow

- All decimal instructions use serial adder.
- First operand is placed into ST; second operand into AB.
- STC specifies which ST byte is to be processed. ABC specifies which AB byte is to be processed.
- Destination bytes replace first operand bytes in ST.
- D contains first operand and destination address.
- IC contains second operand address.
- L1 and L2 specify number of first and second operand bytes, respectively, to be processed.

The data path used for decimal operations consists primarily of ST, AB, and the serial adder (Diagram 3-3, FEMDM). ST contains the first operand, and AB the second. The input byte to the adder A-side is selected from AB under control of the ABC. The input to the B-side of the adder is selected from ST under STC control. The selected bytes are gated to the adder simultaneously.

The serial adder handles the data at a rate of one byte per cycle; i.e., for each two input bytes, one output byte is generated at the SAL. The output byte is gated from SAL to ST under control of the STC, after which the ABC and the STC are decremented and a new cycle is started. Thus, as the operation progresses, the first operand bytes in ST are replaced by the destination bytes.

The number of first and second operand bytes processed depends upon length fields L1 and L2, respectively. The L1 count contained in E(8–11) is decremented once for each byte of first operand that is processed. Similarly, the L2 count in E(12–15) is decremented once for each second operand byte processed.

D contains the address of the first operand, which is also the address of the destination. The initial address in D specifies the doubleword containing the rightmost byte of

the operand. When the STC is decremented to zero, indicating that all first operand bytes in ST have been processed, the contents of ST are stored into main storage. If additional first operand bytes remain in main storage (the L1 count has not stepped to zero), the D-address is decremented by 8, and a fetch of the next operand doubleword is made to ST.

Storage of the destination bytes in ST is controlled by the mark triggers. The mark triggers permit alteration of only those bytes in main storage that belong to the field being processed. There is one mark trigger for each of the eight bytes in ST. As a byte of processed data is gated to ST, the corresponding mark trigger is set, thus designating the byte for main storage.

The IC contains the address of the second operand. (The instruction address is held in the LSWR during execution of SS instructions.) The initial IC address specifies the doubleword containing the rightmost byte of the second operand. When the ABC is decremented to zero, all operand bytes in AB have been processed. If additional second operand bytes remain in main storage (L2 count has not stepped to zero), the IC address is decremented by 8 and a fetch is made of the next second operand doubleword to AB.

This pattern of fetching data, processing via the serial adder, assembling the results in ST, and storing the contents of ST into main storage is continued until either the first or the second operand length field (L1 or L2 count, respectively) has counted below zero. The operation at this point depends upon the individual instruction. If L2 has been exhausted but not L1, some instructions may require extension of the remaining first operand bytes with high-order zeros. On the other hand, if L1 is exhausted before L2, the instruction may test the remaining second operand bytes for presence of significant digits. This test is performed to detect a possible overflow condition and is accomplished by running the excess second operand bytes through the serial adder and sensing nonzeros. In all cases, if both L1 and L2 counts are exhausted, the instruction execution ends after the last destination word is stored into main storage.

Some decimal operations require use of the parallel adder to perform a right-4 or left-4 shift of the entire operand. The "spilled" bits generated during the shift are held in B(64–67).

Program Interruptions

Six program interruptions can occur during execution of decimal instructions. Of the six, only decimal overflow can be masked off; the others are unconditionally taken. If the decimal overflow mask bit [PSW(37)] is a 0, the decimal overflow interruption is ignored; if a 1, it is taken.

The six interruptions and their causes are:

1. Protection. The storage key does not match the protection key in the PSW. The operation is terminated for either a store or a fetch violation.
2. Addressing. An address designates a location outside the available storage for the installed system. The operation is terminated.
3. Specification. A multiplier or a divisor size exceeds 15 digits and sign, or a divisor is equal to or greater than the dividend, or a multiplier is equal to or greater than the multiplicand. The instruction is suppressed.
4. Data. A sign or digit code of an operand specified in the Add, Subtract, Compare, Zero and Add, Multiply, or Divide instruction is incorrect, a multiplicand has insufficient high-order zeros, or the operand fields in these instructions overlap. The operation is terminated before any original data is changed in main storage, except for an invalid digit code which is detected after the first store cycle.
5. Decimal overflow. Execution of the Add, Subtract, or Zero and Add instruction results in an overflow condition. The program interruption occurs only when the decimal-overflow mask [PSW(37)] is a 1. The operation is completed by placing the truncated low-order result into the result field and setting the CC to 3. The sign and low-order digits contained in the result field are the same as they would have been for an infinitely long result field.
6. Decimal divide. The quotient exceeds the specified data field, including division by zero. Division is suppressed. Therefore, the dividend and divisor remain unchanged in storage.

Condition Codes

The results of the Decimal Add, Subtract, Compare, and Zero and Add instructions set the CC as shown in Table 1-9.

Logical Instructions

The logical instructions provide for logical manipulation of data: moving, comparing, bit testing, bit connecting, translating, editing, and shifting. The logical instructions use all five instruction formats and work with both fixed- and variable-field length data. Table 1-10 lists the logical instructions.

For a discussion of the eight-bit zoned character codes, data formats, and operand addressing, refer to Appendix D in this manual.

Table 1-10. Logical Instructions

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
AND	N	54	RX	R1 D2(X2, B2)	AND 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠0
AND	NC	D4	SS	D1(L, B1) D2(B2)	AND 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠0
AND	NI	94	SI	D1(B1) I2	AND immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠0
AND	NR	14	RR	R1 R2	AND 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠0
Compare Logical	CL	55	RX	R1 D2(X2, B2)	Binarily compare 1st opr (in GPR per R1) with 2nd opr (in stg) & set CC according to result. 1. Left to right, byte by byte. 2. Terminate on inequality or end of fields.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLC	D5	SS	D1(L, B1) D2(B2)	Binarily compare 1st opr (in stg) with 2nd opr (in stg) & set CC according to result. 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Terminate on inequality or end of fields.	Prot (F) Adr	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLI	95	SI	D1(B1) I2	Binarily compare 1st opr (in stg) with immediate opr (I2 of inst) & set CC according to result. 1. Left to right. 2. Terminate on inequality or end of fields.	Prot (F) Adr	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLR	15	RR	R1 R2	Binarily compare 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & set CC according to result. 1. Left to right, byte by byte. 2. Terminate on inequality or end of fields.	None	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Edit	ED	DE	SS	D1(L, B1) D2(B2)	Change format of source (2nd opr; in stg) from packed to zoned, edit source under control of pattern (1st opr; in stg), & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr Data	0 : Result = 0 1 : Result < 0 2 : Result > 0
Edit & Mark	EDMK	DF	SS	D1(L, B1) D2(B2)	Change format of source (2nd opr; in stg) from packed to zoned, edit source under control of pattern (1st opr; in stg), place result into 1st opr location, & place location of each 1st significant result digit into GPR1. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr Data	0 : Result = 0 1 : Result < 0 2 : Result > 0

Table 1-10. Logical Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Exclusive OR	X	57	RX	R1 D2(X2, B2)	Exclusive-OR 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠ 0
Exclusive OR	XC	D7	SS	D1(L, B1) D2(B2)	Exclusive-OR 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠ 0
Exclusive OR	XI	97	SI	D1(B1) I2	Exclusive-OR immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠ 0
Exclusive OR	XR	17	RR	R1 R2	Exclusive-OR 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠ 0
Insert Character	IC	43	RX	R1 D2(X2, B2)	Insert 2nd opr (byte; in stg) into bits 24–31 of 1st opr location (in GPR per R1). Remaining bits in GPR are unchanged.	Prot (F) Adr	Unchanged
Load Address	LA	41	RX	R1 D2(X2, B2)	Insert 2nd opr adr into bits 8–31 of GPR specified by R1. 1. Bits 0–7 in GPR are made 0's. 2. 2nd opr is not fetched from stg.	None	Unchanged
Move	MVC	D2	SS	D1(L, B1) D2(B2)	Place 2nd opr (in stg) into 1st opr location (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Move operation can be high or low speed.	Prot (S,F) Adr	Unchanged
Move	MVI	92	SI	D1(B1) I2	Place immediate opr (I2 of inst) into 1st opr location (in stg).	Prot (S) Adr	Unchanged
Move Numerics	MVN	D1	SS	D1(L, B1) D2(B2)	Place numeric portion (low-order 4 bits) of each byte of 2nd opr (in stg) into low-order 4 bits of corresponding byte of 1st opr (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Zones (high-order 4 bits) in both opr's are unchanged. 4. No restriction on overlapping fields.	Prot (S,F) Adr	Unchanged
Move Zones	MVZ	D3	SS	D1(L, B1) D2(B2)	Place zone portion (high-order 4 bits) of each byte of 2nd opr (in stg) into high-order 4 bits of corresponding byte of 1st opr (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Numerics (low-order 4 bits) in both opr's are unchanged. 4. No restriction on overlapping fields.	Prot (S,F) Adr	Unchanged

Table 1-10. Logical Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
OR	O	56	RX	R1 D2(X2, B2)	OR 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠0
OR	OC	D6	SS	D1(L, B1) D2(B2)	OR 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠0
OR	OI	96	SI	D1(B1) I2	OR immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠0
OR	OR	16	RR	R1 R2	OR 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠0
Shift Left Double Logical	SLDL	8D	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd opr adr. 1. R1 must be even adr. 2. High-order bits of 1st opr are shifted out & lost; vacated low-order bits are made 0's.	Spec	Unchanged
Shift Left Single Logical	SLL	89	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd opr adr. High-order bits of 1st opr are shifted out & lost; vacated low-order bits are made 0's.	None	Unchanged
Shift Right Double Logical	SRDL	8C	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd opr adr. 1. R1 must be even adr. 2. Low-order bits of 1st opr are shifted out & lost; vacated high-order bits are made 0's.	Spec	Unchanged
Shift Right Single Logical	SRL	88	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) right number of bit positions specified by low-order 6 bits of 2nd opr adr. Low-order bits of 1st opr are shifted out & lost; vacated high-order bits are made 0's.	None	Unchanged
Store Character	STC	42	RX	R1 D2(X2, B2)	Store bits 24–31 of 1st opr (in GPR per R1) into 2nd opr location (in stg).	Prot (S) Adr	Unchanged
Test Under Mask	TM	91	SI	D1(B1) I2	Set CC according to state of 1st opr bits (in stg) selected by mask bits (I2 of inst). 1. If mask bit = 1, test corresponding 1st opr bit; if mask bit = 0, ignore corresponding 1st opr bit. 2. Character in stg is unchanged.	Prot (F) Adr	0 : Selected bits all 0's (mask is all 0's) 1 : Selected bits mixed 0's & 1's 3 : Selected bits all 1's

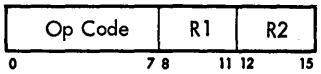
Table 1-10. Logical Instructions (cont)

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Translate	TR	DC	SS	D1(L, B1) D2(B2)	Add 1st opr byte (argument; in stg) to effective 2nd opr adr, use result as stg adr, & place function byte from resulting stg adr into corresponding 1st opr byte location. 1. Effective 2nd opr adr = contents of GPR adr by B2, + D2. 2. LL = number of bytes to be translated. 3. 1st opr bytes are processed left to right.	Prot (S,F) Adr	Unchanged
Translate & Test	TRT	DD	SS	D1(L, B1) D2(B2)	Add 1st opr byte (argument; in stg) to effective 2nd opr adr, use result as stg adr, & test function byte from resulting stg adr. If 0, translate & test next argument byte; if non-0, complete operation by inserting related argument adr into GPR1 & function byte into GPR2. 1. Effective 2nd opr adr = contents of GPR adr by B2, + D2. 2. LL = number of bytes to be translated. 3. 1st opr bytes are processed left to right. 4. Set CC according to ending condition.	Prot (F) Adr	0 : All bytes tested are all 0's 1 : Non-0 byte found before last byte to be tested 2 : Non-0 byte found as last byte to be tested

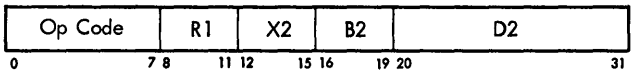
Instruction Formats

Logical instructions use the following five formats:

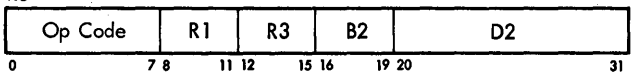
RR



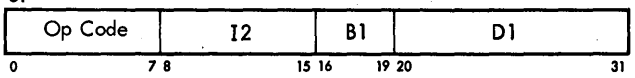
RX



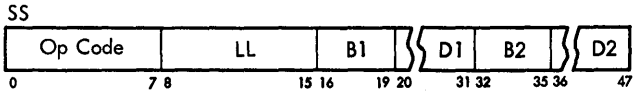
RS



SI



SS



In the RR, RX, and RS formats, the contents of the GPR specified by R1 are called the first operand. In the SI and SS formats, the contents of the GPR specified by B1 are added to the contents of the D1 field to form an address. This address designates the leftmost byte of the first operand field. The number of bytes to the right of this first byte is specified by the LL field in the SS instruction. In the SI format, the operand size is one byte.

In the RR format, the R2 field specifies the GPR containing the second operand. The same GPR may be specified for the first and second operands.

In the RX format, the contents of the GPR's specified by the X2 and B2 fields are added to the contents of the D2 field to form the address of the second operand in main storage.

In the RS format, used for shift operations, the contents of the GPR specified by the B2 field are added to the contents of the D2 field. This sum is not used as an address but the low-order six bits specify the number of bits of the shift. The R3 field is ignored in the shift operations.

In the SI format, the second operand is the eight-bit immediate data field, I2, of the instruction.

In the SS format, the contents of the GPR specified by B2 are added to the contents of the D2 field to form the address of the second operand. The second operand field has the same length as the first operand field.

A 0 in the X2, B1, or B2 field indicates the absence of the corresponding address or shift-amount components. An instruction can specify the same GPR both for address modification and for operand location. Address modification is always completed before operation execution.

Data Flow

Data paths used by the logical instructions are identical to those used by the decimal instructions, with one exception

(Diagram 3-3, FEMDM). For decimal instructions, E(8–15) is divided into L1 and L2 fields. For logical instructions, E(8–15) is one field (LL).

The logical instructions operate on data which may range from 1 to 256 bytes in length. The operands are obtained either from the main storage or from a GPR. Sometimes, the operand may be contained in the instruction itself.

Processing of data in main storage proceeds from the high-order to the low-order address, or from left to right. The initial byte selected for processing may be at either an odd or even main storage address. As a rule, processing of data in a GPR involves the complete register contents. Except for the editing instructions, data is not treated as numbers.

Generally, the operands are treated as eight-bit bytes. In a few cases, the left or right four bits of a byte are treated separately or operands are shifted a bit at a time.

Results replace the first operand, except in the Store Character instruction, where the result replaces the second operand. A variable-length result is never stored outside the field specified by the address and length.

The contents of all GPR's and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the move instructions, and the result locations, GPR1 in the Edit and Mark instruction, and GPR's 1 and 2 in the Translate and Test instruction.

Editing operations provide transformation from packed decimal digits to alphanumeric characters; i.e., editing requires a packed decimal field and generates zoned decimal digits. The digits, signs, and zones are recognized and generated as for decimal arithmetic; all bit configurations are considered valid.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the contents of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function. The list is specified by an initial address, the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The address used to address the list is obtained by adding the argument to the low-order positions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. Where it is known that not all eight-bit argument values will occur, it may be possible to reduce the size of the list.

Use of GPR1 is implied in Edit and Mark and in Translate and Test instructions. A 24-bit address may be placed into this register during these operations. The Translate and Test instruction also implies GPR2. The low-order eight bits of GPR2 may be replaced by a function byte during a translate-and-test operation.

Program Interruptions

Four program interruptions can occur during execution of logical instructions:

1. Protection. The storage key of a result location in main storage does not match the protection key in the PSW. The operation is suppressed on a store violation. The only exceptions are the variable length storage-to-storage operations, which are terminated. The operation is terminated on a fetch violation.
2. Addressing. An address designates a location outside the available storage for the installed system. In most cases, the operation is terminated. The exceptions are the AND, Exclusive-OR, OR, and Move instructions that have the SI format, and the Store Character instruction. These instructions are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of the Load Address instruction is not tested. The address restrictions do not apply to the contents of the D1 and D2 fields, or to the contents of the GPR's specified by X2, B1, and B2.
3. Specification. A full-word operand in a storage-to-register operation is not located on a 32-bit boundary, or an odd register address is specified for a pair of GPR's containing a 64-bit operand. The operation is suppressed.
4. Data. A digit code of the second operand in the Edit or Edit and Mark instruction is invalid. The operation is terminated.

Condition Codes

The results of most logical operations set the CC in the PSW (Table 1-10). The Load Address, Insert Character, Store Character, Translate, and the moving and shift instructions leave this code unchanged.

The CC can be set to reflect five types of results for logical operations. For the Compare Logical instructions, the 0, 1, and 2 states indicate that the first operand is equal to, less than, or greater than the second operand, respectively.

For the logical AND, OR, and Exclusive-OR instructions, the states 0 and 1 indicate a zero or nonzero result field, respectively.

For the Test under Mask instruction, the states 0, 1, and 3 indicate that the selected bits are all-zero, mixed zero and 1, or all-1, respectively.

For the Translate and Test instruction, the states 0, 1, and 2 indicate an all-zero function byte, a nonzero function byte with the operand incompletely tested, or a last function byte nonzero, respectively.

For editing, the states 0, 1, and 2 indicate a zero, less-than zero, or greater-than-zero content of the last result field, respectively.

Branching Instructions

- Branching causes departure from normal instruction sequencing.
- Branch address is introduced as next sequential address.
- Branch address is obtained from GPR or specified as 2nd operand address.
- Branch may be conditional or unconditional.
- Conditional branches (may or may not use branch address):
  Branch on condition
  Branch on count
  Branch on index
- Unconditional branches (always use branch address):
  Branch and link
  Execute
- On branch, normal storage request per IC to fill Q is blocked; branch logic will make request for IC if the branch is unsuccessful and Q needs to be refilled.
- If branch is unsuccessful, Q is refilled if required.

Normally, the CE is controlled by instructions taken in sequential order. That is, an instruction is fetched from a main storage location specified by the instruction address in the IC. The address is then increased by the number of bytes needed to address the next instruction in sequence, and this updated address replaces the old address in the IC. The current instruction is executed, and the same steps are repeated using the updated instruction address to fetch the next instruction.

A departure from the normal instruction sequence occurs when branching is performed. A branch address is introduced as the next instruction address. This branch address may be obtained from one of the GPR's or it may be the second operand address specified by a particular instruction. Depending upon the format and the instruction, branching may be either conditional or unconditional. The conditional branches are branch on condition, branch on count, and branch on index. The unconditional branches are branch and link and execute. Conditional branches may or may not use the branch address. If the branch is successful (that is, the branch is taken), the branch address is used and the storage request issued per the IC during I-Fetch is blocked. If the branch is unsuccessful, the instruction address in the IC is used to fill Q. Unconditional branches are always taken and use the branch address.

Whether a conditional branch is successful depends upon the result of operations concurrent with the branch or preceding the branch. The first case is represented by the branch on count and branch on index instructions. The second case is represented by the branch on condition instructions, which inspect the CC that reflects the result of a previous arithmetic, logical, or I/O operation.

Branching is used to reference a subroutine, to resolve a two-way choice, or to repeat a portion of a program. To save time and increase the speed of the operating program, branching is always considered to be successful unless proven otherwise. (The branch conditions for branch on condition instructions is tested during I-Fetch for a successful or unsuccessful branch, and a D or IC request is issued dependent upon this test.) Therefore, whenever a branch instruction is decoded during I-Fetch, the next instruction address is the branch address located in D. If the branch is found to be unsuccessful (determined during execution of the branch instruction), the instruction address from D is ignored, and the correct instruction address is obtained from the IC.

There are two methods of performing an end-op cycle in the branch operations: normal end op and branch end op. The normal end-op cycle allows decoding of the next instruction format from R and of the instruction address from the IC, and is normally used when ending an operation. Decoding off R is possible because the data placed into the register has become stable by the time the end-op cycle begins. The branch end-op cycle, on the other hand, allows decoding of the next instruction format from the SDBO and of the instruction address from D. This end-op cycle is used when the data, which has been placed into R, is not yet stable and is some halfword other than the last halfword of Q. Decoding from the SDBO saves the time it takes for the data to stabilize in R and the instruction address to stabilize in the IC.

Table 1-11 lists the branching instructions.

Table 1-11. Branching Instructions

Instruction	Mne-monic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Branch & Link	BAL	45	RX	R1 D2(X2, B2)	Store PSW(32–63), link information, into GPR (adr by R1) & branch to location specified by 2nd opr adr. 1. Branch is unconditional. 2. Link information is stored whether or not branch is successful.	Prot (F)†	Unchanged
Branch & Link	BALR	05	RR	R1 R2	Store PSW(32–63), link information, into GPR (adr by R1) & branch to location specified by GPR (adr by R2). 1. Branch is unsuccessful if R2 = 0; use next sequential instr adr. 2. Link information is stored whether or not branch is successful.	Prot (F)†	Unchanged
Branch on Condition	BC	47	RX	M1 D2(X2, B2)	Branch to location specified by 2nd opr adr if state of CC is as specified by M1. 1. Branch is unconditional if M1 is all 1's. 2. Branch is unsuccessful if M1 is all 0's; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Condition	BCR	07	RR	M1 R2	Branch to location specified by GPR (adr by R2) if state of CC is as specified by M1. 1. Branch is unconditional if M1 is all 1's and R2 $\neq$ 0. 2. Branch is unsuccessful if R2 = 0 or if M1 is all 0's; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Count	BCT	46	RX	R1 D2(X2, B2)	Subtract 1 from 1st opr (in GPR per R1); if result $\neq$ 0, branch to location specified by 2nd opr adr. 1. Place result of subtraction into 1st opr location. 2. Branch is unsuccessful if result = 0; use next sequential instr adr. 3. If 1st opr = 1, no branching occurs.	Prot (F)†	Unchanged
Branch on Count	BCTR	06	RR	R1 R2	Subtract 1 from 1st opr (in GPR per R1); if result $\neq$ 0, branch to location specified by GPR (adr by R2). 1. Place result of subtraction into 1st opr location. 2. Branch is unsuccessful if result = 0 or if R2 = 0; use next sequential instr adr. 3. If 1st opr = 1, no branching occurs.	Prot (F)†	Unchanged
Branch on Index High	BXH	86	RS	R1 R3 D2(B2)	Add increment (3rd opr; in GPR per R3) to 1st opr (in GPR per R1), algebraically compare result (index) with comparand (in odd-adr GPR specified by R3 or R3 + 1); if index $>$ comparand, branch to location specified by 2nd opr adr. 1. Place index into 1st opr location. 2. Branch is unsuccessful if index = or $<$ comparand; use next sequential instr adr.	Prot (F)†	Unchanged

Table 1-11. Branching Instructions (cont)

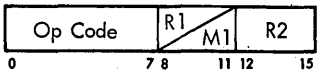
Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Branch on Index Low or Equal	BXLE	87	RS	R1 R3 D2(B2)	Add increment (3rd opr; in GPR per R3) to 1st opr (in GPR per R1), algebraically compare result (index) with comparand (in odd-adr GPR specified by R3 or R3 + 1); if index = or < comparand, branch to location specified by 2nd opr adr. 1. Place index into 1st opr location. 2. Branch is unsuccessful if index > comparand; use next sequential instr adr.	Prot (F)†	Unchanged
Execute	EX	44	RX	R1 D2(X2, B2)	Execute subject instr at location specified by 2nd opr adr. Subject instr may be modified by 1st opr (in GPR per R1) if E(8–11) ≠0. Modification is achieved by OR'ing bits 8–15 of subject instr with bits 24–31 of 1st opr; if R1 = 0, no modification takes place.	Execute Prot (F) Adr Spec	Set by subject instr

†Fetch protected: bit 4 of storage protect set.

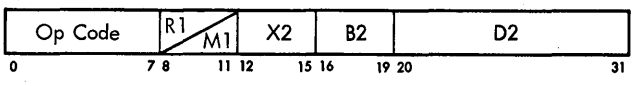
Instruction Formats

Branching instructions use the RR, RX, and RS formats:

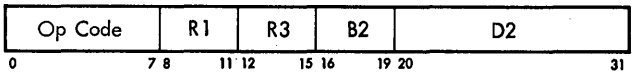
RR



RX



RS



In the formats shown above, bits 8–11 are normally the R1 field that specifies the address of a GPR containing the first operand. In the branch on condition instruction, however, bits 8–11 are designated as M1 and contain mask bits used in conjunction with the PSW CC to determine whether the branch is successful.

In the RR format, the R2 field specifies the address of a GPR that contains the branch address, except when R2 = 0, in which case no branching is to take place.

In the RX format, the contents of the GPR's specified by the X2 and B2 fields are added to the D2 field to form the branch address.

In the RS format, which is used in branch on index operations, the contents of the GPR specified by the B2 field are added to the D2 field to form the branch address. The R3 field specifies the address in LS of an increment value (third operand) which is added to the first operand to determine the index value. R3, if odd, also is the comparand; if R3 is even, R3 + 1 is the comparand.

Data Flow

Diagram 3-4, FEMDM, is a diagram of the basic data flow for the branching instructions. The main functional units used to determine addresses and instructions in the branching operations are Q, R, E, D, and the IC. The secondary functional units, T, AB, parallel adder, STC, and ABC, determine whether the branch is successful when the branch being executed is a conditional branch. The purpose of each functional unit is as follows:

1. Q. Holds the doubleword that contains the instruction addressed by the branch instruction if the branch is successful.
2. R. Contains the instruction to be performed after execution of the branch instruction.
3. E. Contains the branch instruction presently being executed.
4. D. Holds the address of the doubleword which, if the branch is successful, contains the next instruction to be executed.
5. IC. Holds the address of the doubleword which, if the branch is unsuccessful, contains the next instruction to be executed.
6. T. Buffers the operand being tested and operated on.
7. AB. Holds the first operand when added to some other value to determine whether the branch is successful.
8. Parallel adder. Determines whether conditions have been met when a conditional branch is being executed.

9. STC. Allows transfer of last byte of T during an Execute instruction when modifying the subject instruction of the Execute instruction.
10. ABC. Selects data being modified in the subject instruction during an Execute instruction.

Program Interruptions

Four program interruptions can occur during execution of branching instructions:

1. Execute. The subject instruction of an Execute instruction is another Execute instruction. The operation is suppressed.
2. Protection. The branch address of an Execute instruction is protected. The branch-to address of any branch instruction may be fetch-protected. In this case, the PSW key must match the storage key or must be a master key of 0. The operation is suppressed.
3. Addressing. The branch address of an Execute instruction designates an instruction-halfword location outside the available storage area. The operation is suppressed.
4. Specification. The branch address of an Execute instruction is odd. The operation is suppressed.

Condition Codes

The branching instructions leave the CC unchanged, except for the Execute instruction. If the CC is set during the Execute instruction, it is set by the subject instruction.

Status Switching Instructions

- Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions control status of CE.
- Set Storage Key and Insert Storage Key instructions control status of data in main storage.
- Write Direct and Read Direct instructions control status of external element (also transfer data bytes).
- Diagnose instruction controls status of CE.

The status switching instructions can change the status of the CE, the channels, the external device, and the data in main storage. The status of a unit may also be changed by manual intervention and by interruptions (described elsewhere in this manual). The overall status of the CE is determined by the current PSW and associated logic. (For a discussion of the PSW and of the eight CE program states, refer to Section 1 of this chapter.) Any field in the current PSW may be changed directly by the Load PSW instruction, if the CE is in the Supervisor state. Thus, the Load PSW instruction may be used to switch from the Supervisor state to the Problem state, between the Wait and Running states, and between the Masked and Interruptable states. At any time, the Set Program Mask instruction may be used to switch any of the four program mask bits between the Masked and Interruptable states. When in the Supervisor state, the Set System Mask instruction may be used to switch any of the eight system mask bits between the Masked and Interruptable states. The Supervisor Call instruction allows a problem program to switch the CE from the Problem state to the Supervisor state; simultaneously, a byte of information is passed to the supervisor program via the interrupt code of the Supervisor Call old PSW.

Two instructions control the protection status of data in main storage. The Set Storage Key and Insert Storage Key instructions are privileged instructions for controlling the protection status of main storage data in 2048-byte blocks. The Set Storage Key instruction changes the storage protection keys in main storage. The Insert Storage Key instruction fetches the keys from main storage for inspection by the program.

The Write Direct and Read Direct instructions are part of the direct control function. The direct control provides the CE with certain facilities for communication with other CEs and for control of both CEs and IOCEs. These facilities are important to the CE's capability for coordinating the operation of a multisystem.

The Write Direct and Read Direct instructions utilize two sets of lines between CEs and a single set of lines to the IOCEs. Between CEs, 'direct out' lines and 'signal out' lines are used for data communication and control, respectively. 'Signal out' lines to the IOCEs permit the CE to perform certain IOCE control functions. Note that the IOCE is also implemented with the direct control feature to the extent that an IOCE-processor can initiate an external interrupt in a CE. No other direct control functions are available to the IOCEs.

In the CE, the Write Direct instruction can be used to perform eight functions. These eight functions are coded into the I2 field of the instruction and subsequently result in signals on the five 'signal out' lines. The eight functions are:

1. To indicate that a data byte is being sent to a particular CE (on the direct out lines, via a Write Direct instruction) by initiating an external interruption in the receiving CE.
2. To cause a particular CE to perform the external start operation.
3. To cause an automatic logout of a CE.
4. To cause an automatic logout of an IOCE.
5. To cause another CE to reset and go to the Stopped state.
6. To stop an IOCE-processor.

7. To start an IOCE-processor after it has been stopped.
8. To cause an external interrupt of an IOCE-processor.

The Read Direct instruction is used by a CE to accept the data byte from the 'direct in' lines from another CE executing the Write Direct instruction. The Read Direct instruction also initiates an external interruption in the sending CE to inform it that the data byte has been accepted.

For a detailed description of the direct control function, refer to Chapter 6 of the 9020D or 9020E System Introduction Manuals.

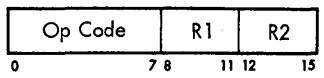
The Diagnose instruction controls the status of the CE. Unlike the Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions that switch the CE's status by changing the current PSW, the Diagnose instruction switches the CE's status by setting control triggers (such as 'defeat interleave' and 'diagnose FLT') through the use of a maintenance control word. Some functions of the Diagnose may also be used by the ATC programmer, when the CE is in state 3, 2, or 1 for various operations (Store DAR, Store PIR, etc.).

Table 1-12 lists the status switching instructions.

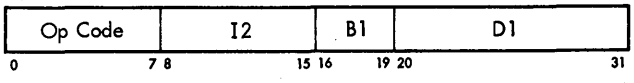
Instruction Formats

Status switching instructions have two formats:

RR



SI



In the RR format, the R1 and R2 fields specify GPR's except when used in the Supervisor Call instruction. The R1 and R2 fields in the Supervisor Call instruction are replaced by an I-field which contains an eight-bit interruption code. In the Set Program Mask instruction, the R2 field is ignored.

In the SI format, the I2 field is ignored for the Load PSW, Set System Mask, and Test and Set instructions. In the Write Direct and Read Direct instructions, the I2 field contains a command and specifies the unit to receive the command. In the Diagnose instruction, the I2 field contains a code for controlling certain maintenance aids. The contents of the GPR specified by the B1 field are added to D1 to form a main storage address of an operand to be fetched by the instruction specified, except for Read Direct. The Read Direct instruction uses the address derived for storing data from an external device. Only one storage address is required in status switching operations. A 0 in the B1 field indicates the absence of the base address component.

Data Flow

- Each status switching instruction has different data flow.
- ST is used by most instructions as buffer before final data transfer.

The status switching instructions transfer data from one unit to another; except for the Insert Storage Key instruction, there is no intermediate processing. Depending on the instruction, the data is obtained from LS, main storage, or the 'direct control bus in' lines and is transferred to either LS, main storage, CE control triggers, or the 'direct control bus out' lines. A generalized data flow is shown in Diagram 3-5, FEMDM. The following is a list of the functional units and their purposes.

1. SCI. Primarily used for 3-cycle fetches of storage operands per D. During the Load PSW instruction, a 3-cycle fetch per the IC is made for the next instruction after the new PSW has been loaded into the CE. For the Set Storage Key, Insert Storage Key, and Test and Set instructions, the SCI performs a four-cycle set-key operation per D, a 3-cycle insert-key operation per D, and a 3-cycle test-and-set operation per D, respectively. During the Read Direct instruction, a three-cycle store operation per D is made.
2. Q. Holds the doubleword containing the instruction being executed. It may also hold the next sequential doubleword if a Q-refill operation occurred during I-Fetch. The Load PSW instruction refills Q with the next instruction regardless of its storage location.
3. R. Contains the instruction to be performed after execution of the status switching instruction.
4. E. Contains the status switching instruction (or the first 16 bits of the instruction) being executed. E(0–7) contains the R code for the Write Direct, Read Direct, and Diagnose instructions, and E(8–15) contains an immediate operand. For the Supervisor Call instruction, E(8–15) contains a supervisor call interruption code. For the Insert Storage Key instruction, E(8–11) contains the address of the GPR into which the protection key is to be inserted.
5. D. Contains the main storage address for storage requests issued during execution of the Set Storage Key, Insert Storage Key, and Read Direct instructions. This register also selects the byte to be used in the Set System Mask and Write Direct instructions.
6. IC. Contains the main storage address of the next instruction during execution of the Load PSW instruction.

Table 1-12. Status Switching Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Diagnose	None	83	SI	D1(B1) I2	Load word designated by stg opr adr into MCW, set or reset certain control triggers, & branch to ROS adr specified by MCW.	Priv Oper Prot (S,F) Adr Spec	Unpredictable
Insert Storage Key	ISK	09	RR	R1 R2	Insert stg protection key for 2048-byte stg block, adr by bits 8–20 of 2nd opr (in GPR per R2), into bits 24–28 of 1st opr (in GPR, per R1). 1. 1st opr: bits 0–23 are unchanged; bits 29–31 are cleared. 2. 2nd opr: bits 0–7 & 21–27 are ignored; bits 28–31 must = 0's. 3. Key is fetched twice because of 2-way interleaving.	Priv Oper Adr Spec	Unchanged
Load PSW	LPSW	82	SI	D1(B1)	Load doubleword stg opr (designated by stg opr adr) into CE, thus replacing current PSW, & branch to new instr sequence. 1. Bits 0–15: system mask, protection key, program state. Bits 16–33: ignored. Bits 34–39: CC, program mask. Bits 40–63: instr adr. 2. If PSW(14) = 1, enter Wait state. 3. If PSW(15) = 1, enter Problem state. 4. Load PSW instr is only instr available for entering Problem or Wait state.	Priv Oper Prot (F) Adr Spec	Set by new PSW bits 34 & 35
Read Direct	RDD	85	SI	D1(B1) I2	Send 'direct control read out' signal & timing signal code (I2; in instr) to external device for about 0.6 usec; store 1 data byte from external device into stg (per stg opr adr) when 'direct control hold in' signal is absent.	Oper Priv Oper Prot (S) Adr	Unchanged
Set Program Mask	SPM	04	RR	R1	Replace CC & program mask (bits 34–39) of current PSW with bits 2–7 of 1st opr (in GPR per R1).	None	Set by opr 1 bits 2 & 3
Set Storage Key	SSK	08	RR	R1 R2	Set stg key (bits 24–28 of 1st opr; in GPR per R1) for 2048-byte stg block (adr by bits 8–20 of 2nd opr; in GPR per R2) into stg protection logic in main storage. 1. 1st opr: bits 0–23 & 29–31 are ignored. 2. 2nd opr: bits 0–7 & 21–27 are ignored; bits 28–31 must = 0's. 3. Key is set twice because of 2-way interleaving.	Priv Oper Adr Spec	Unchanged
Set System Mask	SSM	80	SI	D1(B1)	Replace system mask (bits 0–7) of current PSW with byte from location designated by stg opr adr.	Priv Oper Prot (F) Adr Multisys	Unchanged

Table 1-12. Status Switching Instructions (cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Supervisor Call	SVC	0A	RR	I	Cause supervisor call interruption; replace old PSW(24–31) with I-field (bits 8–15) of instr, providing interruption code. 1. Clear PSW(16–23). 2. Store old PSW at stg location 32 (decimal). 3. Fetch new PSW from stg location 96 (decimal).	None	Unchanged
Write Direct	WRD	84	SI	D1(B1) I2	Send 'direct control write out' signal & timing signal code (I2; in instr) to external device for about 0.8 usec; make 1 data byte from stg (per stg opr adr) available to external device until next WRD is executed.	Oper Priv Oper Prot (F) Adr	Unchanged

7. AB. Buffers operands for the serial adder and the parallel adder. During I-Fetch of the Set Program Mask, Set Storage Key, and Insert Storage Key instructions, the first operand is placed here. During the Set System Mask and Diagnose instructions, doublewords from storage are received here.
8. ST. Buffers operands for the serial adder and the parallel adder. Data is received here from main storage for the Load PSW instruction and from LS for the Set Storage Key and Insert Storage Key instructions. Data is stored from here into main storage during the Read Direct instruction and from LS during the Insert Storage Key instruction. The Load PSW, Set Program Mask and Set System Mask instructions cause all or part of the PSW register to be changed per ST. The Diagnose instruction causes the MCW register, scan counters, and ROSAR to be changed per ST.
9. ABC and STC. Controls selection of data from and placement of data into AB and ST, respectively. Also, during the Read Direct instruction, STC sets a mark trigger.
10. Mark. Identifies the byte to be used by main storage during the Read Direct instruction. All mark triggers are set during the Set Storage Key instruction by a ROS micro-order.
11. F. Buffers the storage key before it is placed into main storage during the Set Storage Key instruction and after it is taken from main storage during the Insert Storage Key instruction. This register also buffers data received from another CE during the Read Direct instruction.
12. G. Buffers a byte of data being sent to another CE when executing a Write Direct instruction.
13. PSW register. Contains a portion of the current PSW. All or part of the PSW register contents is changed directly by the Load PSW, Set Program Mask and Set System Mask, instructions. Because the Supervisor Call, Write Direct, Read Direct, and Diagnose instructions may cause an interruption after being executed, they may indirectly change all of the PSW register contents.
14. MCW register. Controls CE or channel diagnostic functions during and after execution of the Diagnose instruction.
15. Parallel adder. Provides the data transfer path between AB, ST, D, and the IC. Adds 8 to the IC and D for address updating. Subtracts 8 from A during the Set Storage Key and Insert Storage Key instructions so that a re-entrant loop may be constructed. Calculates IC - D + 7 for the address store compare tests made during Read Direct instruction.
16. Serial adder. Provides the data transfer path from AB to ST and G. During the Insert Storage Key instruction, the contents of F are logically OR'ed with the contents of T via the serial adder.
17. LS. Contains operands required by the Set Program Mask, Set Storage Key, and Insert Storage Key instructions. Only the Insert Storage Key instruction transfers data into LS.

Program Interruptions

Four program interruptions can occur during execution of status switching instructions:

1. Privileged Operation. Occurs if a Load PSW, Set System Mask, Set Storage Key, Insert Storage Key, Write Direct, Read Direct, or Diagnose instruction is encountered while the CE is in the Problem state. The operation is suppressed.
2. Protection. Occurs if the storage key of the location designated by the instruction does not match the protection key in the current PSW.
3. Addressing. Occurs if an address designates a location outside the available main storage.
4. Specification. Occurs if (1) the operand address of a Load PSW or Diagnose instruction does not have 0's in

the three low-order bit positions, or (2) the block address specified by the Set Storage Key or Insert Storage Key instruction does not have 0's in the four low-order bit positions. The operation is suppressed.

Condition Codes

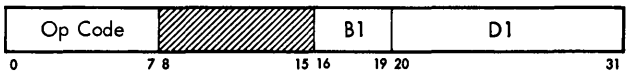
Three status switching instructions affect the condition code: Load PSW, in which the CC is set by new PSW(34,35); Set Program Mask, in which the CC is set by bits 2 and 3 of the first operand; and Diagnose, in which the CC is unpredictable (except during diagnose storage logout when a CC of one indicates a timeout on storage interface). The remaining status switching instructions leave the CC unchanged.

Input/Output Instructions

The CE has five I/O instructions: Start I/O, Test I/O, Halt I/O, Test Channel, and Set PCI (Table 1-13).

Instruction Format

The five I/O instructions use the SI format:



Bits 8–15 are ignored. The base plus the displacement determines the channel and I/O unit address: bits 16–23 of

Table 1-13. Input/Output Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Halt I/O	HIO	9E	SI	D1(B1)	Terminate current I/O operation at selected channel & I/O unit. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr.	Priv Oper	0 : Interruption in channel 1 : CSW stored 2 : Halted 3 : Unavailable
Set PCI	SPCI	9B	SI	D1(B1)	Sets the PCI flag bit in the subchannel controlling the addressed device and causes the subchannel to request an I/O interruption.	Priv Oper	0 : Subchannel not working. No action has been caused. 1 : CSW Stored 2 : PCI Flag Set 3 : Invalid I/O Address Format.
Start I/O	SIO	9C	SI	D1(B1)	Select specified I/O unit & initiate channel command to that unit. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr. 3. CAW, which specifies address of 1st CCW, is fetched from location 72 (48, hex).	Priv Oper	0 : Available 1 : CSW stored 2 : Working 3 : Unavailable
Test Channel	TCH	9F	SI	D1(B1)	Test state of selected channel & set CC accordingly. 1. D(13–15) is channel adr. 2. D(16–23) is ignored. 3. State of channel is not affected.	Priv Oper	0 : Available 1 : CSW ready 2 : Working 3 : Unavailable
Test I/O	TIO	9D	SI	D1(B1)	Clear interruption condition in addressed channel or associated I/O units, & set CC according to status of addressed channel & I/O units. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr. 3. CSW is stored at location 64 (40, hex) if: a. I/O unit or control unit contains pending interruption. b. I/O unit or control unit is executing previous operation, or there is pending channel-end/control unit-end for another I/O unit. c. I/O unit or its control unit detects machine error.	Priv Oper	0 : Available 1 : CSW stored 2 : Working 3 : Unavailable

the sum are the channel address (of which only bits 21–23 are valid), and bits 24–31 of the sum are the I/O unit address.

Data Flow

The CE operation for the five I/O instructions is identical. The data flow path used for the I/O instructions is shown in Diagram 3-6, FEMDM. Listed below are the main functional units used to perform the instructions.

1. D. Contains the channel and unit addresses at the beginning of execution.
2. L.PSBAR and P.PSBAR. Combine to form the one address (the IOCEs PSBA) sent to the IOCE.
3. B and S. Hold the channel and unit addresses temporarily while they are being gated to T.
4. F. Used when combining logical and physical PSBARs in assembly of IOCEs PSBA.
5. Serial adder. Data path for gating B-bytes to S.
6. T. Holds the one-word format that is shipped to the IOCE.
7. External register. Gates the one-word format onto the control (external) bus for transfer to the selected IOCE.

Program Interruption

The only program interruption that may occur for an I/O instruction is the privileged-operation interruption. It occurs if the CE is in any state other then Supervisor. The instruction is suppressed before the channel is selected. The CSW, the CC in the PSW, and the state of the addressed channel and of the I/O unit remain unchanged. The interruption code in the program old PSW(20–31) is 0000 00000010.

Condition Codes

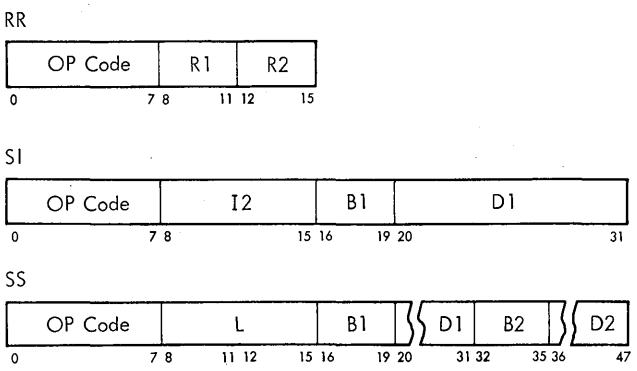
When the CE is released from an I/O instruction, one of four CC's is set into the CC register of the CE and becomes a part of the current PSW. This CC is the result of tests by the IOCE, the channel, or the I/O unit and indicates various conditions that exist in the channel, the control unit, or the I/O unit. The CC's for the five I/O instructions are summarized in Table 1-13. For a detailed discussion of the setting of the CCs, refer to the 9020 D/E Principles of Operation.

Multiple Computing Element Instructions

This section discusses the ten instructions (Table 1-14) that make up the multiple computing element instruction set. The need for this instruction set develops when multiple computing elements must operate simultaneously, without conflict, in a multiple element shared storage environment such as the 9020D/E system. These instructions provide for system configuration, storage assignment, and preferential storage area assignment in the 9020D/E system.

Instruction Formats

The multiple computing element instructions use three formats:



In the RR format, R1 specifies the address of a GPR into which data is loaded or which contains the first part of data to be gated to selected system elements. The R2 field specifies the address of a GPR which contains the second part of the data to be gated to selected system elements and/or a selection mask.

In the SI format, the contents of the GPR specified by the B1 field are added to the contents of the D1 field to form the address at which the first operand is to be fetched or stored. The I2 field, when used, contains data to be gated to selected system elements.

In the SS format, the contents of the GPR specified by the B1 field are added to the D1 field to form the address of the leftmost byte of the destination field. The contents of the GPR specified by the B2 field are added to the D2 field to form the address of the leftmost byte of the source field. The L1 and L2 fields are combined to form one L field, which contains the number of words to be moved from one location to another.

Data Flow

The data flow paths for these instructions are shown in Diagram 3-7, FEMDM. Some of the main functional units and their functions are listed below.

1. Local Storage. Holds the operands and/or masks of many of the instructions.

Table 1-14. Multiple Computing Element Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Delay	DLY	0B	RR	N	Provides variable delay determined by value of N times 256 microseconds.	None	Unchanged
Load Identity	LI	0C	RR	R1	Loads CE identity into GPR specified by R1 field.	None	Unchanged
Load PS Base Address	LPSB	A1	SI	D1(B1)	The preferential-storage base address register is loaded from the operand location.	Priv Oper Prot (F) Adr Spec	Unchanged
Store PS Base Address	SPSB	A0	SI	D1(B1)	The contents of the logical and physical preferential-storage base address registers are stored at the operand location.	Priv Oper Prot (s) Adr Spec	Unchanged
Set Configuration	SCON	01	RR	R1,R2	Select elements indicated in select mask and set selected elements CCRs according to the configuration mask.	Spec Priv Oper	0 = all selected elements accepted CCR info with correct parity 2 = one or more elements failed to accept the CCR
Set Address Translator	SATR	0D	RR	R1,R2	Select elements indicated in select mask and set selected elements ATRs according to the storage element assignment mask.	Spec Priv Oper	0 = all elements accepted mask 1 = some element not configured. 2 = some element detected parity error in mask. 3 = selection mask is all 0s
Insert ATR	IATR	0E	RR	R1,R2	Content of ATR is placed in a pair of registers specified by R1 and R2 fields.	None	Unchanged
Move Word	MVW	D8	SS	D1(B1) D2(B2)	The second operand field is moved to the first operand location.	Prot (F) (S) Adr Spec	Unchanged
Start I/O Processor	SIOP	9A	SI	12	The operand address and a protection key are made available to the designated IOCE-Processor.	Priv Oper	0 = PSW loaded in IOCE. 1 = Invalid PSW. 3 = Not Operational
Test and Set	TS	93	SI	D1(B1)	Test high-order bit (0) of the addressed storage byte. Set CC according to state of tested bit, and set addressed byte back into storage as all 1's.	Prot (S,F) Adr.	0: High-order bit = 0 1: High-order bit = 1

2. Control (External) Bus. Provides the data path for the transfer of data from a unit in one element to a unit in another element.
3. External register. Holds data that is gated onto the control bus.
4. Select register. Holds the select bits of elements involved in the execution of the instruction. The bits of this register become the actual 'select' lines to the selected elements.
5. 9020 Out Bus. The bus that provides the data path for gating the contents of the multiple computing element registers onto the local storage out bus and into ST for processing.
6. CCR. Its contents determine the configuration of the system and set up communication paths between the elements of the system. Set, via the control bus, with information obtained from local storage.
7. ATR. Holds the addressing information that determines the physical configuration of storage elements. Set directly from ST.
8. ABC and STC. Control selection of data from, and placement of data into, AB and ST, respectively.
9. Serial Adder. Provides the data transfer path from AB to ST. The adder latches are tested in many instructions to determine the value of some byte or partial byte.
10. F. Buffers data during some operations. Used as a counter in conjunction with the serial adder in many operations.

Program Interruptions

Four program interruptions can occur during execution of the special 9020 instructions. In all cases the operation is suppressed.

1. Privileged Operation. Occurs if an LPSB, SPSB, SATR, SIOP, or SCON is encountered while the CE is in the problem state.
2. Protection. Occurs if the storage key of the location designated by an LPSB, SPSB, or MVW does not match the protection key in the PSW.
3. Addressing. Occurs if an address designates a location outside the available storage for the particular installation, or outside the configured storage or storage assigned by the storage address translator for a particular CE.
4. Specification. Occurs if:
   a. The operand address of an LPSB, MVW, or SPSB does not have the two low-order bits both 0.
   b. Bit positions 8-19 of the operand addressed by LPSB do not specify a location within a storage element provided in the ATR or do specify a DE in a 9020E system.
   c. Execution of SCON or SATR is attempted by a CE whose own SCON bit is off in its CCR or whose state bits are set to 1 or 2.
   d. A configuration mask has all SCON-field bits 0's.
   e. A selection mask has an IOCE selection bit set when two or more CE communications bits are set in the configuration mask.
   f. A storage assignment mask references a storage module not available to a particular installation, assigns an SE module to a position reserved for a DE module in the particular installation, or assigns a DE module to other than positions 6-10 on a 9020E system.
   g. A SIOP does not select an IOCE, selects more than one IOCE or the first operand address does not reference a doubleword storage location or a key of F (hex) was given, or bit 15 was not 0.

Condition Codes

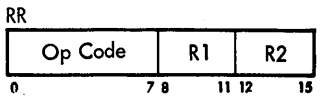
Four multiple computing element instructions (SCON, SATR, TS, and SIOP) set the condition code. These instructions operate with other system elements, and the condition codes that result indicate the status of the affected elements. The resulting condition codes are shown in Table 1-14.

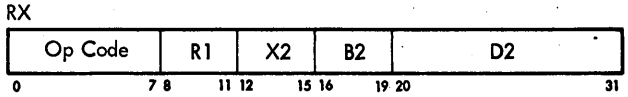
Display Instructions

The display instructions (Table 1-15) provide for data management of display storage, for processing of radar and weather line input data, and for storage paging in the environment of the 9020E Display Channel Processor. The main objective of the display instructions is to provide an up-to-date display image for each of the PVDs attached to the 9020E system.

Instruction Formats

The display instructions use two instruction formats:





In the RX format, used by the Load Chain instruction, the first operand is the contents of the even GPR of an

Table 1-15. Display Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruption	Condition Code
Load Chain	LC	52	RX	R1, X2, B2, D2	Moves first operand to third operand location and second operand to first operand location.	Prot (F) Adr	0 = Bit 31 of 2nd operand is 0 3 = Bit 31 of 2nd operand is 1.
Convert and Sort Symbols	CSS	02	RR	R1, R2	Submits an input stream of primary radar or beacon data to one PVDs geographic and sterile area filters and stores an output word into one of sixteen sort bins for each input target that passes all filters.	Prot (F) (S) Adr Spec	0 = processing complete, no data stored 1 = processing complete data stored 2 = page boundary encountered 3 = invalid beacon format
Convert Weather Lines	CVWL	03	RR	R1, R2	Submits an input stream of weather line coordinates to one PVDs geographic and sterile area filters and stores an output doubleword in the PVDs refresh memory for each input line that passes all filters.	Prot (F) (S) Adr Spec	Unchanged
Repack Symbols	RPSB	0F	RR	none	Assembles an updated display image for one-sixteenth of a PVDs area.	Prot (F) (S) Adr Spec	0 = processing complete 2 = page boundary encountered

even/odd pair of GPRs specified by the R1 field. The contents of the odd GPR of this pair is the third operand. The contents of the GPRs specified by the X2 and B2 fields are added to the content of the D2 field to form the address of the second operand.

In the RR format, used by the three other display instructions, the R1 field is used to identify the type of input to be processed by CSS. CVWL and RPSB ignore the R1 field.

The R2 field is used by CSS and CVWL to address the GPR containing the address of the input data; it is ignored by RPSB. The address of the input stream is contained in GPR9, so the R2 field must always contain the value 9.

Data Flow

The display instructions accomplish most of their tasks by moving data from one main storage location to another, reformatting it in the process in some cases. The main data flow paths are shown in Diagram 3-8, FEMDM. The main functional units used, and their purposes, are listed below:

1. AB. Holds data to be moved during instruction execution. Also used in boundary calculations.
2. IC. Addresses main storage area for the source data (data to be moved) on most operations.
3. D. Addresses main storage area for the destination of the data (location to which the data is moved).
4. ST. Used in boundary calculations. Is the data path for data from the GPRs to all other registers. Is also a data path to the M-register.
5. LM. Holds data read from main storage (data which requires reformatting before being stored into the new location).
6. Mixer. Reformats data in LM by gating it, according to conditions present, to the proper positions in XY.
7. XY. Holds reformatted data from the mixer and gates the data onto SDBI for storage at a new location in main storage.
8. K. Holds data for later use and accumulates total counts of the data being moved.
9. Local Storage. Contains control information for the display instructions. Provides work area for certain operations.
10. Serial Adder. Provides a data path from AB to ST. Rearranges hex characters within a byte where needed. Compares bytes from different registers where required.
11. N. Holds current descriptor during that portion of RPSB execution.
12. Parallel Adder. Performs boundary calculations and total count additions; provides main data path from AB to ST.

Program Interruptions

Three program interruptions can occur during execution of the display instructions.

1. Protection. Occurs if the storage key of an accessed location does not match the protection key in the PSW and either the storage key or the protection key is not 0.
2. Addressing. Occurs if an address designates a location outside the available storage for the particular installation, or outside the configured storage or storage assigned by the storage address translator for a particular CE.
3. Specification. Occurs if (1) the second operand address of the LC instruction is not on a doubleword boundary, (2) in CSS, CVWL, or RPSB, the data chain address contained in bytes 5-7 of the doubleword following a 512-byte page is not on a doubleword boundary, or (3) if either the old refresh memory address or the new refresh memory address is not on a doubleword boundary in RPSB.

Condition Codes

Three display instructions set the condition code: LC, CSS, and RPSB. These instructions set codes which indicate the status of their execution or the reason for their termination. The resulting condition codes are shown in Table 1-15.

MAINTENANCE FACILITIES

Maintenance and troubleshooting of the CE is facilitated by the use of Logout, the Diagnose instruction, ROS tests, fault locating tests (FLT's), ripple test, diagnostic programs, and marginal checking. The CE contains special hardware facilities for each of these functions. A brief description of these functions and the associated hardware is provided in the following paragraphs.

Logout

Logout is the process of storing the status of the CE control panel indicators into the logout area of the PSA. Special circuitry called scan logic is incorporated into the CE to perform the logout. The scan logic provides a separate means of control and special data paths, so that a logout may be obtained even though normal CE logic is disabled by a malfunction.

A number of checks are made during logout to insure the accuracy of the logout data. For example, any of the following checks cause the CE to hardstop:

1. Log ROS Check: CE accesses a ROS word not in the logout routine.
2. Log ADR Check: CE attempts to store data outside the logout area.
3. PSBAR Parity Check: PSA location may not be correct.
4. PSBAR Alternate Check: An attempt has been made to step PSBAR a second time.
5. Storage Address or Storage Data Check.

In normal operation, the logout occurs automatically when a machine check occurs (CHECK CONTROL switch in PROC position and machine-check interruptions not masked off). A logout can also be initiated manually via a pushbutton on the CE control panel.

Diagnose Instruction

The Diagnose instruction provides for a number of special operations to be performed by a program running in states 3, 2, or 1; these include logout of an SE, reset check, reading the DAR and PIR, setting the DARM, and others. For maintenance purposes, however, the Diagnose instruction provides a much broader range of operations for diagnostic programs running in state 0. These include defeating of storage interleaving, running portions of FLT tests, forcing certain incorrect parity conditions to test error detection circuitry, entering a microprogram at any desired point, and a special DE wrap function used in the 9020E system. The Diagnose instruction utilizes a register called the maintenance control word (MCW) register. MCW information in the Diagnose (operand address field) is set into the MCW register during execution of the Diagnose instruction. Bits in the MCW register then cause the desired operation to be performed by the hardware.

Other Diagnose functions are not mentioned here, but a complete explanation is provided in Chapter 4.

ROS Tests and FLTs

ROS tests and FLTs utilize the same special scan logic as Logout, together with the MCW register. These tests are read into the CE from tapes which are computer-generated from source data used in the manufacture of the CE ROS planes and logic. The ROS tests check each bit of each ROS word. The FLTs scan data into the CE hardware, advance the clock if necessary, and log the data out again. In this

manner, the circuitry may be tested without performing any instructions. Fault detection is performed at the logic level (i.e., at the level of the individual AND, OR, and Invert blocks making up the CE logic).

Microprogram Diagnostic

A microprogram, starting at ROS address FAA, may be manually initiated to check out most CE registers by means of the DATA keys. Normal parity checking is used to detect errors.

Ripple Tests

Ripple tests provide for the testing of local and main storage by rippling data from the data switches throughout the desired storage. These tests provide a quick check of a large portion of CE and storage hardware (SCI, CE to SE interface, etc.). The ripple tests provide a quick confidence test of CE and SE operation and, used in combination with the parity check indicators, a means of identifying a failing major area within them.

Diagnostic Programs

Diagnostic programs provide a major means of maintaining and confidence-testing the CE. These programs use normal instructions and hardware, except for the Diagnose, as mentioned previously. Separate documentation is provided with the diagnostic programs.

Marginal Checking

Marginal checking is the process of testing the CE while it is operating under nonstandard voltages or clock frequency. This procedure provides a means of detecting circuits which have become voltage- or frequency-sensitive, often before they fail in normal operation.

POWER

All CE power, from the theory of operations standpoint, is covered in the 9020D/E Power Controls and Distribution Manual. Chapter 1 covers the system aspects of power; Chapter 2 covers CE power specifically. Power maintenance information is provided in the CE FEMM.

This chapter is divided into seven sections:

Section 1, Timing and Clock Control.
Section 2, Read-Only Storage.
Section 3, Data and Control Registers.
Section 4, Local Storage.
Section 5, Serial and Parallel Adders.
Section 6, Status and Control Triggers.
Section 7, Storage Control Interface.

Each functional unit is described separately, in regard to operation, operational timing, and functional application. Supporting the descriptions are simplified, positive-logic upper-level diagrams, flowcharts, and timing charts.

SECTION 1. TIMING AND CLOCK CONTROL

The CE operates with a basic clock cycle of 200 ns, i.e., a 5-mHz clock frequency. Each cycle comprises clock and not-clock portions, used for data transfer and logic functions, respectively. The 200-ns clock cycle is divided into twenty 10-ns intervals for intracycle timing.

CLOCK SIGNAL GENERATORS

The CE uses a 5-mHz gated delay-line oscillator to provide the basic clock signal. The gated delay-line oscillator can be inhibited by the SCI during a storage request (Diagram 4-1, FEMDM). The oscillator is inhibited within one cycle after the SCI issues a 'select' signal to storage. When the 'accept' signal is received from storage, the clock is restarted after a pre-adjusted time delay which synchronizes the CE with the arrival of data from storage.

The gated delay-line oscillator reduces the time required to restart the clock. It is not necessary to wait a full oscillator cycle (200 ns) to restart the clock as would be the case with a continuously running oscillator. The delay-line oscillator provides stable clock signals immediately upon restart. The negative-going signal into the oscillator not only starts the oscillator but becomes part of the first output cycle.

A continuously running crystal-controlled oscillator, (Diagram 4-2, FEMDM) provides a 5-mHz reference frequency for adjusting the 5-mHz delay-line oscillator frequency. A comparator circuit (Diagram 4-3, FEMDM) mixes the two signals and provides an output that is the absolute difference between the two. When the two signals are within 1 kHz of each other, the output of the comparator circuit is a null, indicating that the delay-line oscillator frequency is within 0.02% of the crystal-controlled oscillator frequency.

For test purposes (CE in state 0), the clock signal generator may be operated at a higher output frequency (5.128 mHz), thereby shortening the clock cycle period by 2.5% (from 200 ns to 195 ns). The higher frequency is obtained from both the CE and the reference oscillators by setting the FREQUENCY ALTERATION switch to the down position and the TEST switch to the TEST position. Both switches are located on the CE control panel.

CLOCK TIMING

- Triggers are set and reset at clock time.
- Latches are set and reset at not-clock time.
- Logic operations normally occur at not-clock time; subsequent data transfers occur at following clock time.
- Symmetrical and unsymmetrical clock signals are used, depending on logic function.
- Twenty 10-ns delay intervals provide for intracycle timing.

Throughout the CE, trigger and latch logic is used for all data-handling and control functions. Although implemented with the same logic components (ANDs, ORs, and inverters), triggers (by definition) are set or reset at clock time, whereas latches (by definition) are set or reset at not-clock time. In general, the data registers consist of triggers, and all intermediate logic units (such as adders, incrementer/decrementers, and decoders) consist of latches. Control logic consists of both triggers and latches. The CE design provides for all intra-CE data manipulation to be done by register-to-latch-to-register (trigger-to-latch-to-trigger) sequences, in lieu of direct register-to-register (trigger-to-trigger) transfer. Continuous availability of stable data results from the overlapping of the trigger and latch set/reset states (Figure 2-1). Note that the data (e.g., "A" in the figure) is stable in either a trigger or a latch at any one

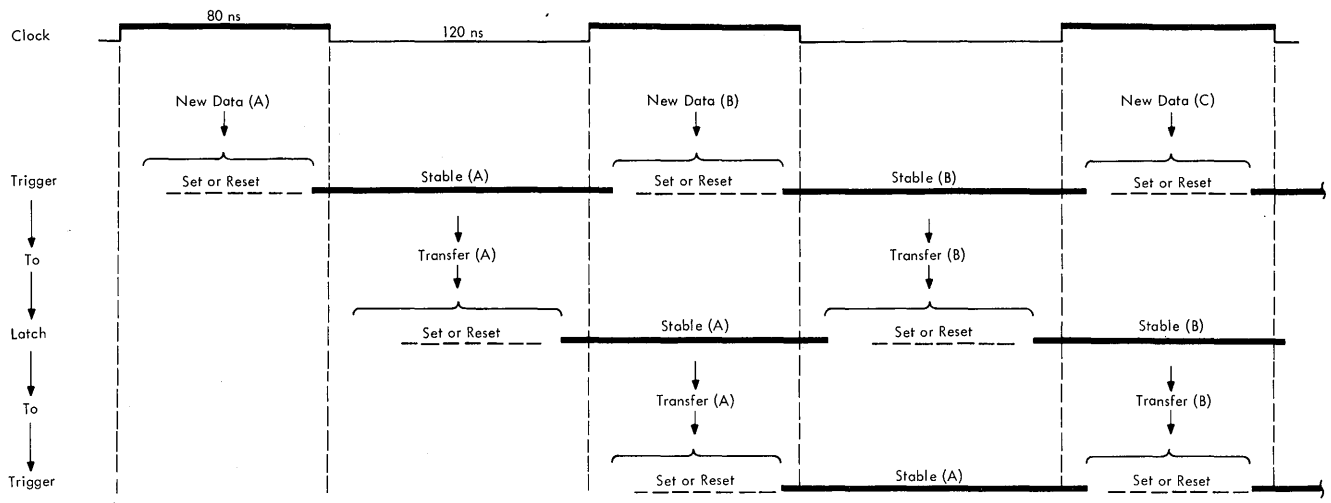


Figure 2-1. Trigger and Latch Data Relationship

time during the indicated two clock cycles, but that the trigger and latch are available for new data ("B") in the next clock cycle.

Each logic component within the CE introduces some degree of signal delay which must be considered in the CE operation. All logic blocks with inversion introduce between 3 ns and 20 ns of delay, with 10 ns as the average. (Some special circuits introduce either 30 ns or 700 ns.) All logic blocks without inversion introduce less than 3 ns of delay. In addition, approximately 10-ns to 12-ns delay is created by every 6 feet of signal transmission line.

Because of these delays, the clock signal is converted from a 5-mHz symmetrical signal to a 5-mHz unsymmetrical signal, with an 80-ns clock time and a 120-ns not-clock time. This unsymmetrical signal provides the needed extra time for logic operations during not-clock time and still leaves sufficient time for trigger input at clock time.

Many logic blocks are used in the parallel adder at clock time, thus causing excess delay. To overcome this delay, the unsymmetrical clock signal is extended to a symmetrical clock signal with a 100-ns clock time and a 100-ns not-clock time; this conversion is made within the parallel adder. Some of the parity checking and sign propagating circuits are timed by the unsymmetrical clock signals. Primary concern in clock timing is to provide stable information at sampling time.

Oscillators A through E (Diagram 4-1, FEMDM) set up controls to stop, start, or control the clock. These signals are used in advance of clock time so that the controls are stable when they are needed.

With most of the CE logic blocks introducing 10 ns of delay, the 200-ns clock cycle period is divided into twenty 10-ns delay intervals to provide timing within a clock cycle (Figure 2-2). These delay intervals are called "B" time or "P" time and are relative to the start of the clock cycle. "B" time refers to symmetrical clock signals; "P" time refers to unsymmetrical clock signals. The delayed clock signals are created by series inverters (each inverter introducing one 10-ns delay interval) or by time delays (e.g., B0, B1, B2, ... , and P0-1, P0, P1, P2, ...).

CLOCK CONTROL AND SIGNAL DISTRIBUTION

- Manual, SCI, and ROS operations and errors affect clock signal availability.
- Two clock signals, one symmetrical and one unsymmetrical, are distributed throughout logic gates.
- Time delays unskew and synchronize clock signals within and between sections of logic.

The availability of the clock signals to the CE processing logic is controlled by the clock-stopping logic (Diagram 4-1). The SCI has the ability to inhibit the clock signal generator to account for the cabling delays to the storage elements (Diagram 4-1). During maintenance operations (such as scan, log-out, single-cycle, DE wrap, and diagnose log-out storage), the clock signals may be stopped or permitted to run intermittently.

The 'pass pulse' trigger (Diagram 4-1) provides clock signal distribution control during both normal (continual) and single-cycle operations. A start, load, or reset operation sets the 'pass pulse' trigger and permits the clock signals to be passed on to the logic. When in the single-cycle mode, the 'block' trigger (set by the same operations) resets the 'pass pulse' trigger and blocks the clock signals before the next clock cycle (unless the SCI holds the clock on).

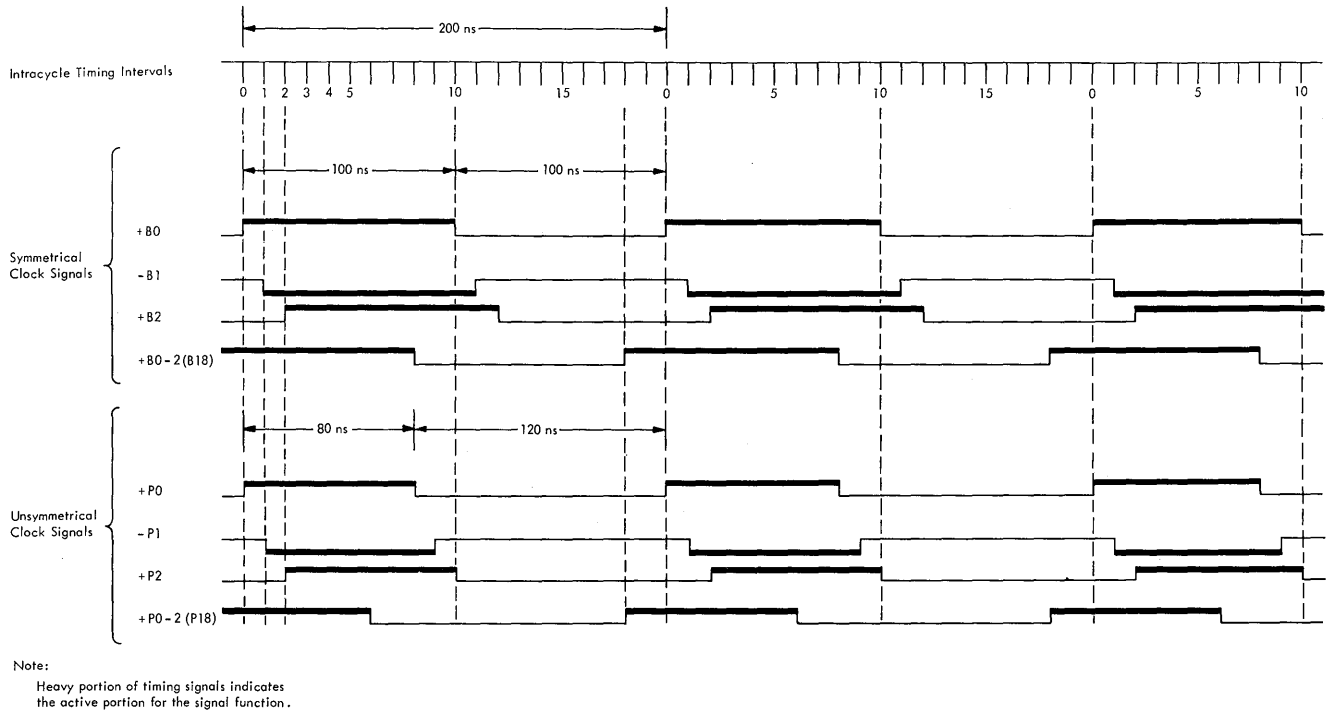


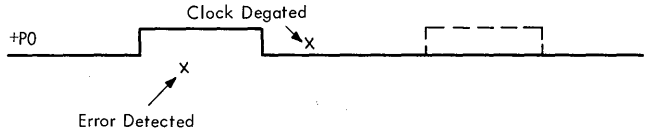
Figure 2-2. Typical Clock Signals

The 'stop clock' trigger (Diagram 4-1) provides SCI control of the clock signal distribution. If the SCI cannot process a CE storage request immediately (due to a busy SE or DE), the SCI sets the 'stop clock' trigger and blocks the clock signals. When the request can be completed, the SCI generates the 'BCU cleanup' signal, which resets the 'stop clock' trigger.

During certain operations, the ROS microprogram may stop the CE clock signal distribution for one or two cycles ('STOP1' or 'STOP2' micro-orders). The 'stop clock ROS' trigger provides this control from the bit configuration of ROS word bits 45 and 46 (Diagram 4-1).

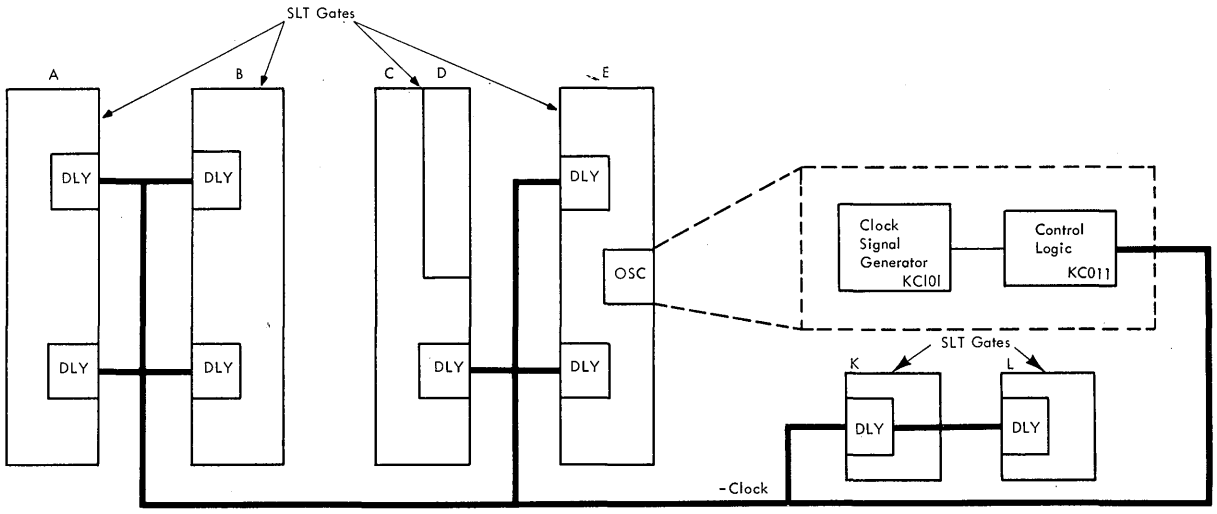
The distribution of clock signals to portions of the ROS logic is stopped while the CE is in the Wait state (Diagram 4-1). The Wait state is entered if PSW (14) = 1 at end op. An interruption or entering the stop loop removes the wait state block of the ROS signals. The 'time clock step' trigger coming on also removes the Wait state block of ROS (Diagram 6-20, FEMDM). In this case, ROS branches to a microprogram which steps the time clock, and upon completion of this, if there has been no change in PSW (14), the CE re-enters the Wait state.

Clock signal distribution to the CE processing logic is stopped when the 'error' trigger is set and the CE CHECK CONTROL switch is not in the DSBL (disable) position. Error detection occurs during clock time, and the clock signals cease with the next not-clock time:

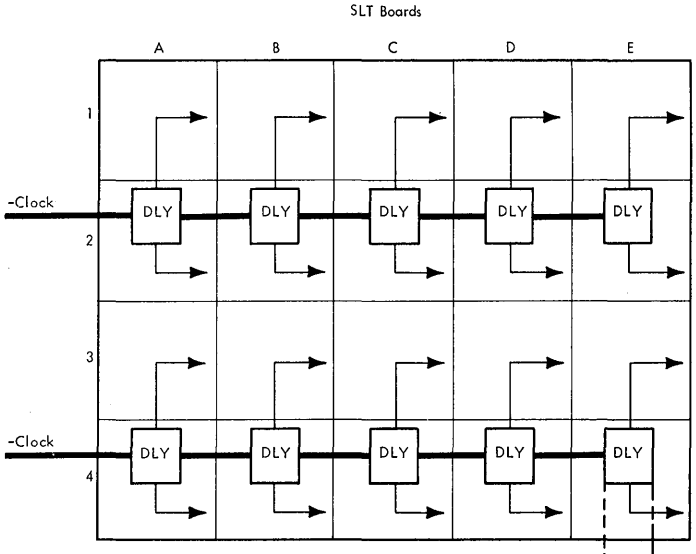


Necessary information for error analysis is thus held in latches and triggers to be examined directly or to be stored by a logout operation. Note that the DLY (delay) units allow the clock to finish the cycle started by the last clock signal. The clock may be restarted by the CE CHECK CONTROL switch, by internal circuits, or by resetting the 'error' trigger (inputs to clock inhibit, Diagram 4-1).

The clock signal development and distribution concept is shown in Figure 2-3; note that it is not representative of any logic gate. Figure 2-3 (A) shows how the master 'clock' signal is distributed to the gates. Figure 2-3 (B) shows distribution within a typical gate, with a separate delay logic for every two SLT boards. The adjustable time delay shown in Figure 2-3 (C) allows for unskewing of the clock signals, i.e., aligning all 'P0' signals. The 'special sync for P0 B0' signal is used to adjust the time delays so that all 'P0' signals occur simultaneously.



A. Clock Distribution

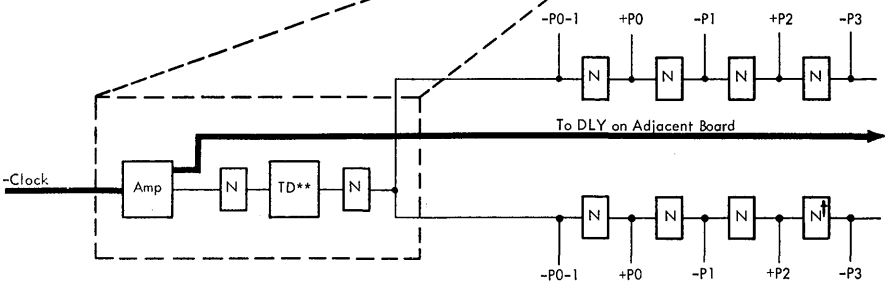


B. Typical 5 X 4 Gate

Note: Total delay includes 10 ns per 6 feet of wire plus adjustable time delay.

** TD is adjusted to synchronize each P0 with 'special sync for P0 B0' signal shown on ALD ZA001. Each P0 must coincide with every other P0 in any SLT gate.

† Number of inverters varies from gate to gate, depending on need for particular signal.



C. Unskewing Delay Logic

Figure 2-3. Clock Signal Development and Distribution

The read-only storage (ROS) is a device containing a permanently recorded microprogram used to control CE operations. The microprogram is in the form of 100-bit micro-instructions (ROS words), each of which has a unique predetermined bit pattern. The ROS words can be read out as required, but a physical modification is necessary to change the stored information. When decoded, the bits of the ROS word condition gates whose outputs perform the necessary functions to execute an operation. Thus, ROS eliminates the need for most complex instruction decoders and sequencing networks and introduces a flexibility to machine design not previously available in control hardware. This flexibility allows changes to be made to control circuits (for special features) by replacing printed circuit sheets in ROS.

CAPACITIVE READ-ONLY STORAGE ARRAY

The capacitive read-only storage (CROS) array consists of 2816 100-bit ROS words, which are addressed by a 12-bit ROS address register (ROSAR). The array consists of 16 planes, each of which is divided into 4 quarter planes. Each quarter plane has one array driver energizing 1 of 22 select lines. Each select line causes two ROS words to be read out. To address a particular drive line from ROSAR, bits 0–3 select a plane, bits 4 and 10 select a quarter plane, and bits 5–9 energize one select line. Bit 11 of ROSAR selects one of the two ROS words (upper or lower) read out each cycle.

CROS Electrical Theory

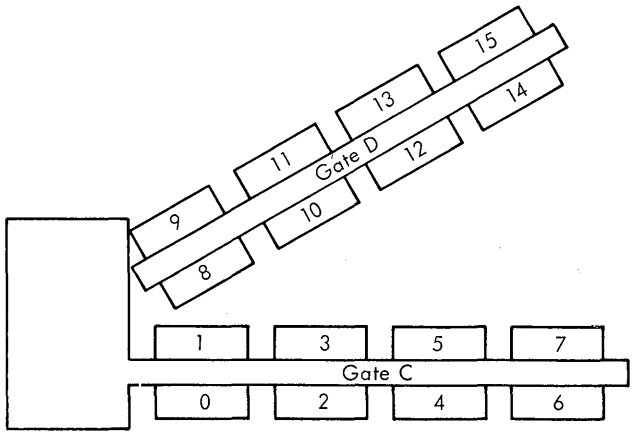
The CROS operates on the presence or absence of a capacitor between a drive line and a sense line. Only one driver at a time may be energized. In the example shown in Figure 2-4, when driver 1 is energized, an impulse is coupled through capacitor C1 to differential sense amplifier D, producing the "D" bit. The same drive results in inputs to differential sense amplifiers A, B, and C, but the polarity is reversed and no bits are generated. To equalize the capacitive load (impedance) to all sense amplifiers, a balance line is provided with each driver and is allowed to "float".

Note: Because the balance line function was found to be unnecessary, later machines do not use the balance line, although it is still printed on the ROS planes.

Some unwanted capacitive coupling exists in this type of matrix. In Figure 2-4, when driver 1 is energized, C1 couples the voltage shift to sense line D, C2 couples the voltage shift to drive line 2, and C3 couples the voltage shift to sense line B. This unwanted signal is very low because it passes through three elements in cascade. The threshold of the sense amplifier is designed so that the low signal is rejected while the desired signal is amplified.

CROS Planes

The 2816 words of ROS are stored in 16 planes. Each plane contains the capacitors, drive lines, balance lines, and sense lines for 176 100-bit ROS words. The drive and balance lines are independent, whereas the sense lines feed common sense amplifiers. Planes 0–7 are on gate C, and planes 8–15 are on gate D:



Drive and Balance Lines (Bit Plates)

The drive and balance lines are photo-etched from a sheet of copper that is bonded to epoxy glass (Figure 2-5). The resulting epoxy sheet with copper drive and balance lines is called a bit plate. A separate bit plate controls the bit configuration for each CROS plane.

Tabs at the top and bottom of the bit plate are used for electrical connections to the drive and balance lines. The top tabs connect the drive and balance lines to terminating resistors. The bottom tabs connect the drive lines to the drive circuits.

Four holes in the bit plate align the bit plate to the sense plane. The two outer holes snap over locating studs in the sense plane, and the inner two holes provide clearance for the center studs.

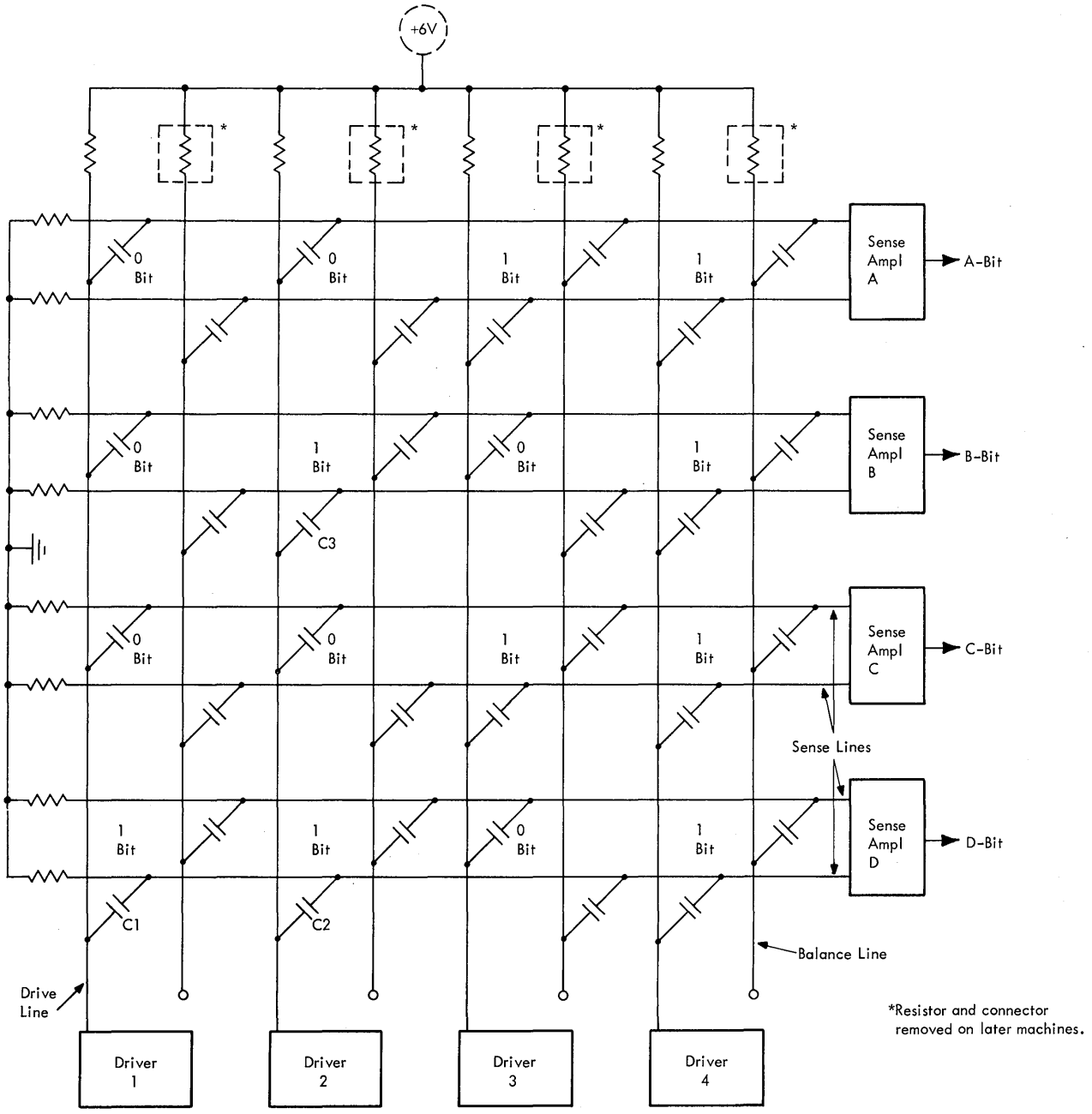


Figure 2-4. Basic 4 x 4 CROS Matrix

Sense Lines

- 200 pairs of sense lines are in each CROS plane.
- A pair of sense lines carries signal for one ROS-word bit position.
- 200 pairs of sense lines read out two 100-bit ROS words simultaneously.

The sense lines are photo-etched into copper-covered epoxy-glass plates (Figure 2-6). The sense-line plates are permanently mounted to the array gates. Electrical connections from the sense lines to the terminating resistors and sense amplifiers are made with low-temperature solder.

There are 200 pairs of sense lines in each CROS plane. Two sense lines are required to read out one bit position of the ROS word. One drive line simultaneously reads out two 100-bit ROS words, which use 200 pairs of sense lines.

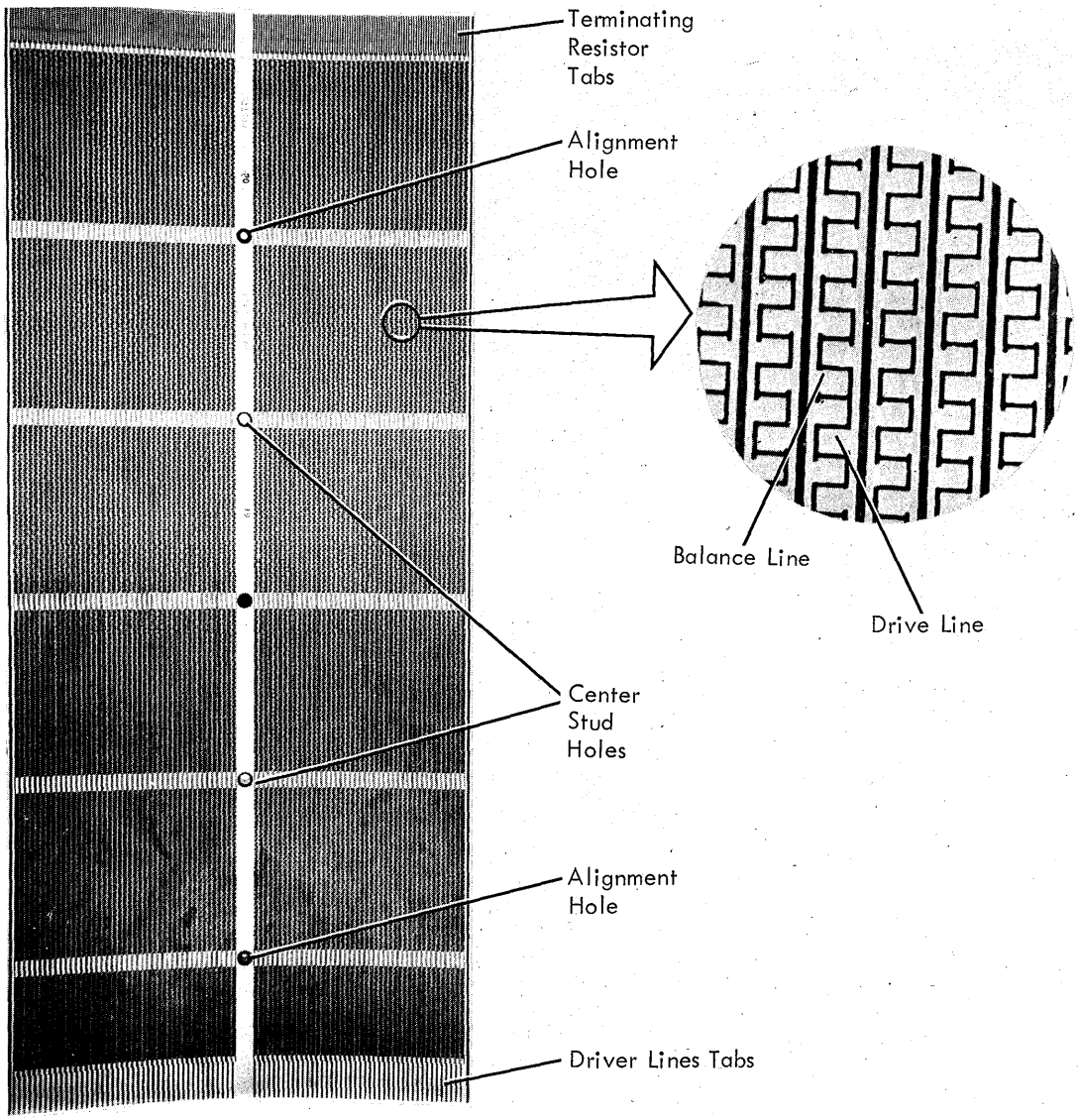


Figure 2-5. Bit Plate

Figure 2-7 shows the layout of the sense lines in the ROS planes. The top pair of sense lines is bit 0 of the upper word. The next lower pair of sense lines is bit 0 of the lower word. This order continues to the bottom pair of sense lines, which is bit 99 of the lower word. The upper and lower words are read out simultaneously. Each sense line is terminated through a resistor to ground. Note the distribution of sense lines through the planes to the differential sense amplifiers. The sense lines through the planes on both sides of a gate are tied together for each bit. The pair of sense lines from each gate is then ORed in the sense amplifier for each bit. Because only one plane has an active drive line for a given ROS address, the sense amplifier receives only one input signal.

Bit Capacitors

The bit capacitors are formed by sandwiching a sheet of Mylar† between the bit plate and the sense lines (Figure 2-8). Pressure plates hold these pieces firmly together. The Mylar is the dielectric, and the drive, balance, and sense lines become the plates of the capacitors.

Tabs on the drive and balance lines increase the size of the capacitors to form the bit configuration. The effective capacitive coupling of a drive line to a sense amplifier is equal to C1 minus C2. The size of this effective capacitor is approximately 0.5 pf.

†Trademark of E. I. duPont de Nemours & Co. (Inc.)

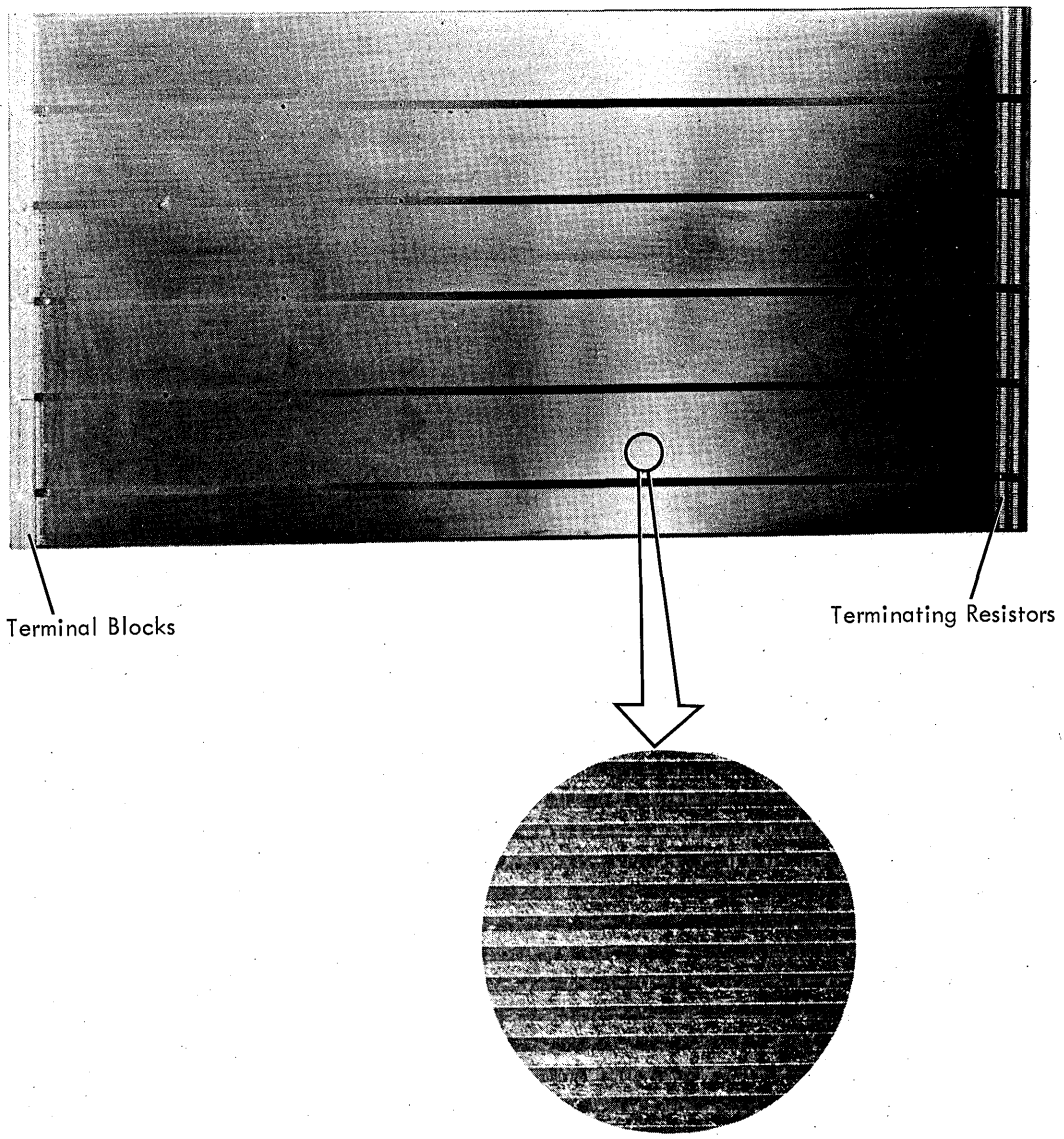


Figure 2-6. Sense Lines

The bit configuration within a CROS plane is controlled by the bit plate. Therefore, the ROS word can be changed by replacing the bit plate that contains the word.

Physical Package

A CROS plane consists of a sandwich comprising the sense line board, a dielectric sheet, and a bit plate. These pieces are held firmly together by pressure plates (Figure 2-9).

A pressure plate, with a neoprene pad, fits over each group of capacitors in the plane. The plates are loosely connected to a pressure frame that is bolted to the gate. Adjusting screws in the frame squeeze the pressure plate against the bit plate. Because the sense lines are on a rigidly mounted board, the pressure plate holds the bit-capacitor sandwich firmly together.

Electrical connections to the bit plate are also made through pressure connections.

ROS ADDRESSING

ROS word addresses are assembled in the 12-bit read-only storage address register (ROSAR). Each ROS word contains the basic address of the next ROS word. The basic address may be modified by machine operation or by error conditions.

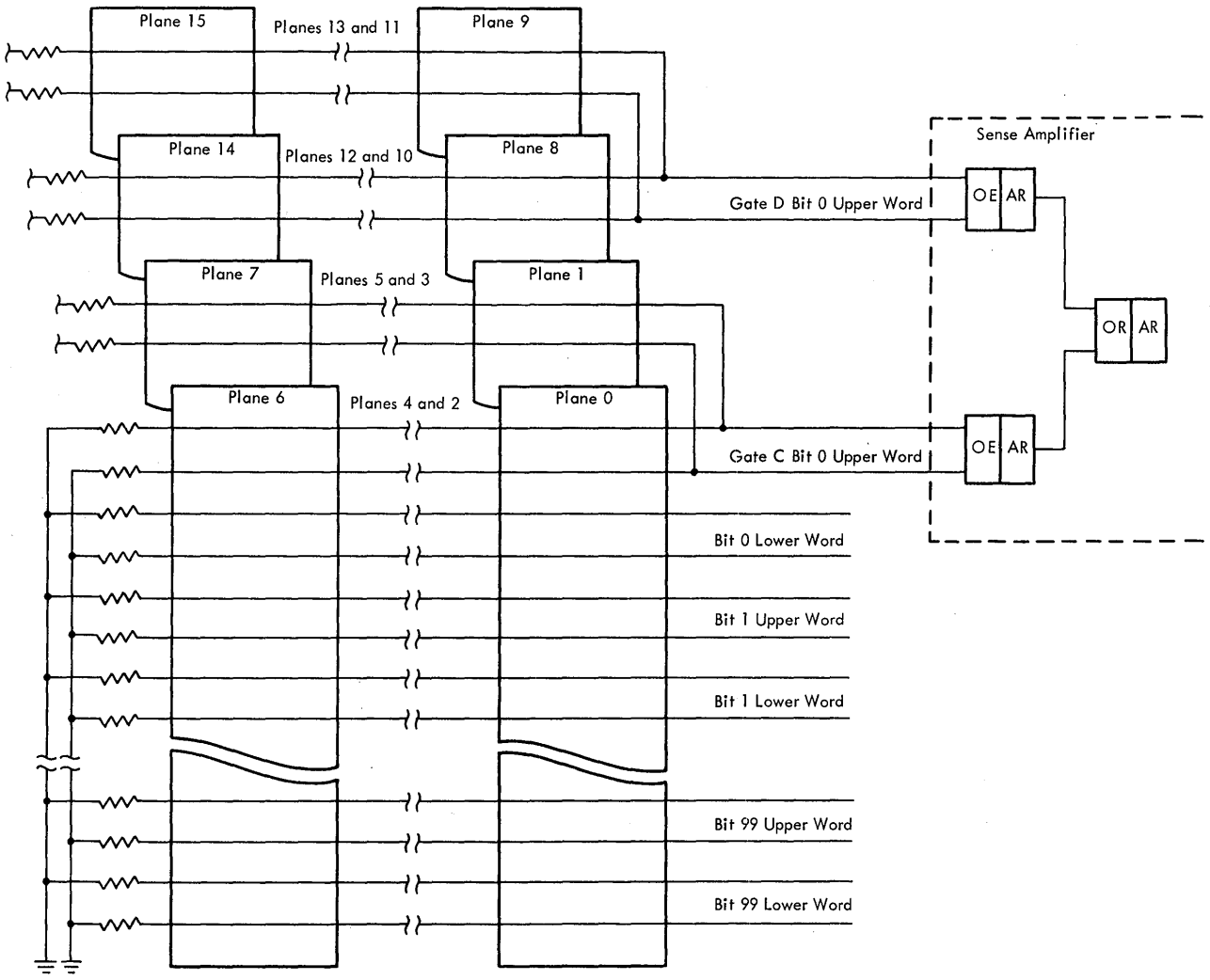


Figure 2-7. Sense Line Layout

Read-Only Storage Address Register

- ROSAR(0–11) supplies 12-bit address that selects next ROS word.
- Overriding branch and manual ROS operations force new address to ROSAR.

The ROSAR, a 12-position (labeled 0–11) latch register, supplies the address to select the next ROS word. The configuration of the ROSAR contents (address) is controlled by the NA, K, and J control fields of the present ROS word. A new address is available in ROSAR 50 ns after each P2 clock time. Although each ROS word contains the address of the next word to be accessed, address modifications can result by satisfying data-dependent branch conditions. These data conditions are stable at ROSAR(0–10) by P2 + 30 ns and at ROSAR(11) by P2 + 50 ns. The gate at ROSAR is a P4 clock pulse. To prevent late branching, the output of all ROSAR bits must be stable by P2 + 60 ns; at approximately P0 minus 30 ns, the ROS sense latches are sampled. This sequence is repeated every machine cycle.

The ROSAR bit positions can be divided arbitrarily into four groups, according to their inputs. These groups are: (1) ROSAR(0–5), which normally receives only the six high-order bits of the base address; (2) ROSAR(6–9), which can receive the four low-order bits of the base address and/or the output of the X-branch decoder; (3) ROSAR(10), which can receive data from either the X-branch decoder or the Y-branch decorder; and (4) ROSAR(11), which can receive data from either the X-branch decoder or the Z-branch decoder. [Note the overlap between the base address and X-branches on ROSAR(6–9).] An overriding branch, however, affects all positions of the ROSAR.

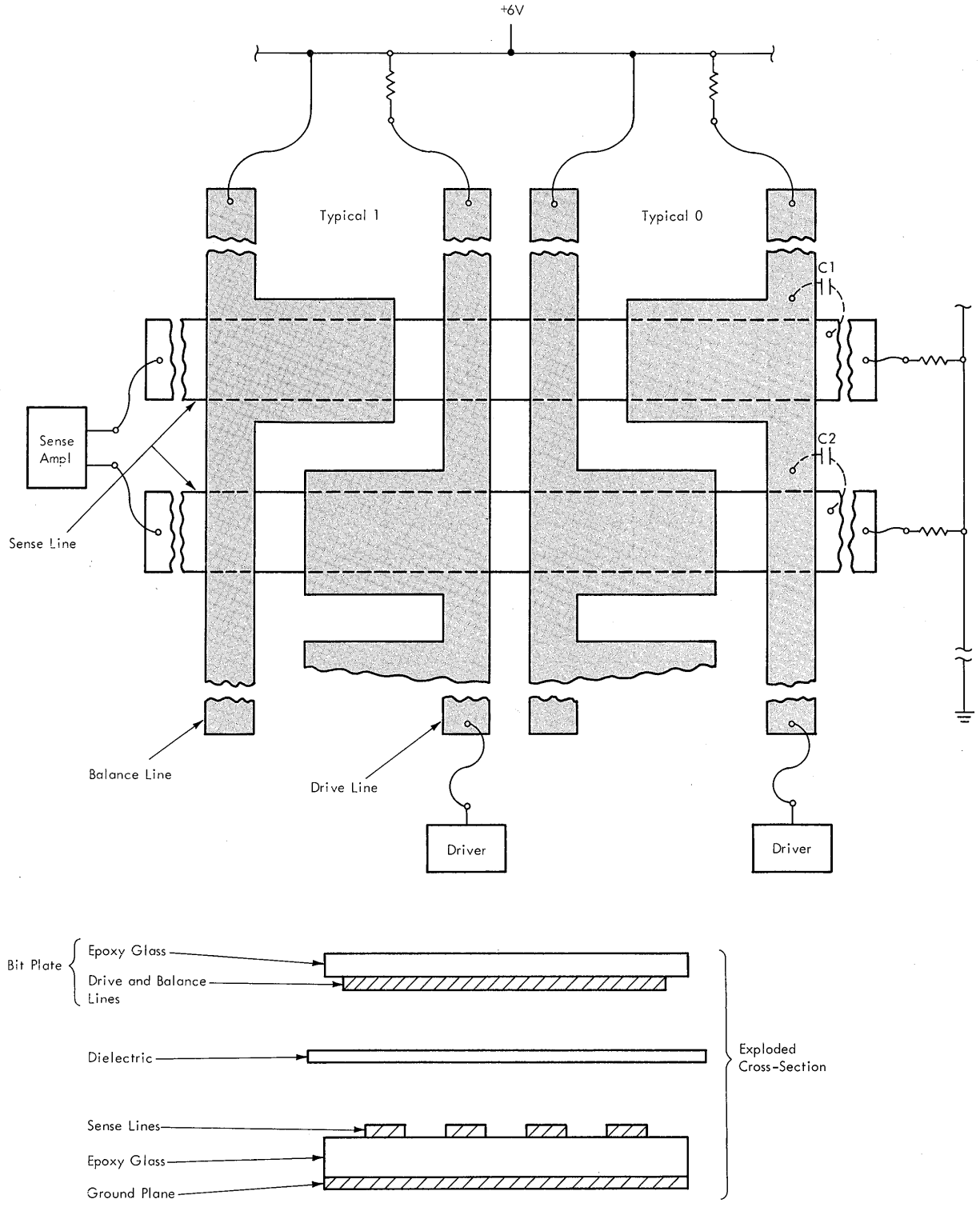


Figure 2-8. Bit Capacitors

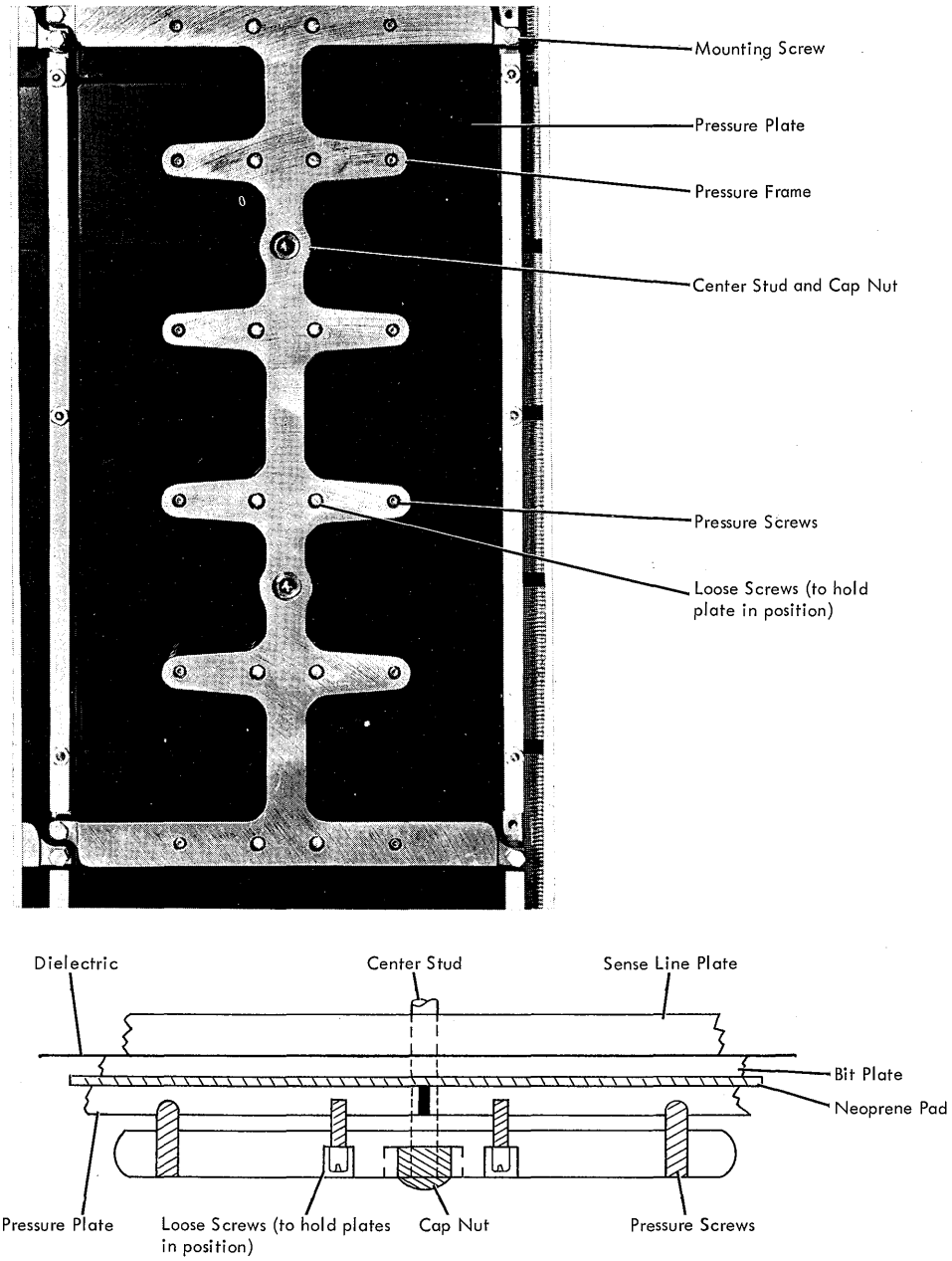


Figure 2-9. CROS Plane Pressure Mounting Assembly

The following micro-orders can cause an overriding branch: 'T→RAR', 'EXCEP', and 'SPEC'. If control field K contains the 'T→RAR' micro-order, the base address is inhibited and T(40–51) is transferred to ROSAR(0–11). This branch is unconditional. An 'EXCEP' micro-order inhibits the base address if an interruption is pending. The source of the interruption provides the branch address. When an 'SPEC' micro-order is specified, a specification program interruption forces the branch address into the ROSAR. In addition, a local store write operation is blocked.

Certain FLT operations force a new address into ROSAR (Diagram 4-101, FEMDM).

Two manual operations cause the contents of the ADDRESS switches to be forced into ROSAR: depressing the ROS TRANSFER pushbutton and activating the REPEAT ROS ADDRESS switch. The operation of these controls is described in Section 1 of Chapter 4.

ROSAR(0–5)

ROSAR(0–5) (Diagram 4-101, FEMDM) can be set from one of three sources: ROS sense latch bits 47–52, T(40–45), or ADDRESS keys 8–13. Normally, positions 0–5 receive the five high-order bits of the base address

from the NA control field of the current ROS word contained in the ROS sense latches. However, if control field K contains an overriding branch and the branch condition is met, or if an FLT operation is in progress and certain conditions are present, or if certain manual operations are being performed, the base address is inhibited from entering the ROSAR and ROSAR(0–5) is set from the ST bus or the ADDRESS switches.

ROSAR(6–9)

ROSAR(6–9) (Diagram 4-102, FEMDM), in addition to receiving the low-order bits of the base address and the overriding branch address, can be set individually by X-branches (functional branches) specified by control field J (bits 62–68) of the ROS word. The control field J micro-orders that specify X-branches are listed as J96 to J125 on ALD A6231, which also shows the ROSAR bits that are set under specific conditions for each X-branch micro-order. When an X-branch is executed, the high-order positions [ROSAR(0–5)] remain unchanged; i.e., they still contain the high-order bits of the base address. The base address bit positions corresponding to the bits affected by the X-branch must be set to zero when an X-branch micro-order is given. For example, if ROSAR(6) can be set to a 1 by a certain micro-order, ROS sense latch bit 53 must be 0 in the ROS word that contains the branch.

ROSAR(10)

ROSAR(10) (Diagram 4-103, FEMDM) can be set by an X-branch, an overriding branch, and/or a Y-branch. The base address, however, has no effect on this bit. Also, because an overriding branch and a Y-branch are both decoded from bits in control field K (bits 57–61) of the ROS word, only one of the two branches can be executed at a time; they cannot be executed together. That is, if control field K specifies an overriding branch, a Y-branch cannot be specified, and vice versa.

However, a Y-branch and an X-branch can be executed together because they are functions of micro-orders in separate control fields (K and J). The result of ROSAR(10) is as follows: if neither the X-branch condition nor the Y-branch condition is met, ROSAR(10) remains a 0; if either or both of these conditions are met, ROSAR(10) is set to a 1. The Y-branch micro-orders that affect ROSAR(10) are listed in the K field on ALD A6241 in the same manner as that of X-branches.

ROSAR(11)

ROSAR(11) (Diagram 4-104, FEMDM) can be set by an X-branch, an overriding branch, or a Z-branch. The base address, just as for ROSAR(10), has no effect on ROSAR(11). Because an X-branch and a Z-branch are both decoded from bits in control field J (bits 62–68), only one of the two branches can be executed at one time. ALD A6231 lists the Z-branch micro-orders.

However, a Y-branch and a Z-branch can be executed together. The effects on ROSAR(10,11) are as follows: if neither condition is met, both ROSAR bits remain a 0; if either condition is met, the associated ROSAR bit is set to 1; if both conditions are met, both bits are set to 1's.

If control field K contains an overriding branch and control field J contains a Z-branch, and if the overriding-branch condition is met, the result of the Z-branch is inhibited and ROSAR(11) is set as specified by the overriding branch.

ROSAR(0–10) Decoding

ROSAR decode logic decodes the address in ROSAR(0–10) to select 1 of 1408 array drive lines. ROSAR(0–4,10) is decoded into 1 of 64 drive lines; ROSAR(5–9) is decoded into 1 of 22 select lines; and ROSAR(0) selects gate C or D. One select line and one drive line then select 1 of the 1408 array drive lines. See Diagram 4-105, FEMDM, for decode flow. ROSAR(11) is decoded to select one of the two ROS words read out each cycle. (If bit 11 = 1, the lower word is selected; if bit 11 = 0, the upper word is selected.)

Strobed Drive Lines

A ROSAR address selects 1 of 64 strobed drive lines by decoding ROSAR(0–4,10) as shown in A of Diagram 4-105, FEMDM. The decoded address is a gate-drive signal to the array drivers. Each strobed drive line controls the array drivers for one CROS quarter plane.

Decoding is accomplished in two levels. In the first level, bits 3, 4, and 10 are decoded to activate 1 of 8 lines, and bits 0, 1, and 2 activate 1 of 8 lines. The outputs of the two first-level decoders are then combined with a gate-drive signal to activate 1 of 64 drive lines.

Select Lines

One of 44 select lines is activated by decoding ROSAR(0,5–9) (B of Diagram 4-105). Twenty-two of these select lines are connected to gate C, and 22 to gate D. ROSAR(0) is decoded to select the gates, and ROSAR(5–9) is decoded to activate a select line within a gate. Although ROSAR(5–9) can be decoded 32 different ways, only the first 22 combinations are considered valid addresses; the other 10 combinations are not tested. If an

illegal bit combination is entered into these bit positions, no select line is activated. Illegal addresses are addresses in which ROSAR(5,6) = 11.

Decoding is accomplished in two levels. In the first level, bits 7, 8, and 9 are decoded to activate 1 of 7 lines, and bits 0, 5, and 6 are combined with a 'gate word select' signal (clock P2 delayed) to activate 1 of 6 lines. (Note that when bits 5 and 6 = 11, no signals are developed from the first-level decoder.) The outputs of the two first-level decoders are then combined in the second-level decoder to activate one of the 44 select lines.

Array Drivers

There are 1408 array drivers in ROS, 704 on gate C, planes 0–7, and 704 on gate D, planes 8–15. Diagram 4-105, C, shows how the drive-line signals are developed and distributed. Each array driver is an AND that ANDs 1 of 64 drive lines with 1 of 22 select lines (details are shown in Diagram 4-106, FEMDM). The ANDs are single transistors; the drive lines condition the emitters while the select lines control the bases. Voltage is supplied to the collectors through the array drive-line-terminating resistors.

Sense Amplifiers

The sense amplifiers increase the voltage difference between paired 1 and 0 sense lines. The first stage of the sense amplifiers (D of Diagram 4-105, FEMDM) consists of two differential amplifiers, one for gate C sense lines and one for gate D sense lines. Because only one array driver is active for a machine cycle, the sense lines of only one gate carry a signal during a machine cycle. The first-stage differential amplifier increases the voltage difference between paired sense lines and sends this signal to the second-stage amplifier. The second-stage further amplifies the signal and transmits it to the ROS sense latches.

ROSAR(11) Function

Each cycle, 200 bits are sent to the sense latches. ROSAR(11) divides this information into two 100-bit words. If ROSAR(11) = 1, the lower word is selected; if ROSAR(11) = 0, the upper word is selected.

ROS DATA FLOW

ROS word data is transferred from the sense amplifiers to the sense latches. A portion of the word is immediately decoded, while other portions step through registers and latches to provide a delay so that the data is available at the desired time. The ROS word data flow is shown in F of Diagram 4-105.

ROS Sense Latches

The 100 ROS sense latches hold the ROS word for decoding and for setting ROSDR at P0 of the next machine cycle. The sense latches are set by strobing either the upper or lower word sense amplifier outputs and are reset approximately 120 ns after P0 of the machine cycle (E of Diagram 4-105).

ROS Data Register and ROSDR Latches

The ROS data register (ROSDR) holds fields A through H, and M, N, P, Q, and W of the ROS word for use in the next machine cycle. Fields H, and M, N, P, and Q are decoded directly from the ROSDR to control LS and the adders, respectively. Fields A–G and W, however, are further delayed by holding them in the ROSDR latches. These fields are used for register ingating.

The ROSDR latches allow a ROS word to control certain gates during the register set time of the next cycle. For example, one ROS word may contain the micro-instruction: add the contents of T and A, and store the answer into A. The ROS word adds the contents of the registers on one machine cycle and stores the sum from the parallel-adder-out bus at register set time of the next cycle.

Diagram 4-107, FEMDM, is a simplified diagram of ROSDR. In this diagram, each main division of ROSDR is represented by a single bit position. At clock P0-1 of each machine cycle, ROSDR is reset. At not-clock P0-1, the contents of the ROSDR (bits 6–36) are sent to the ROSDR latches. At clock P0, the contents of the sense latches are transferred to the ROSDR. The output of the ROSDR latches (containing the previous ROS word) and the output of ROSDR (38–42, 69–77, and 78–84) are then decoded to perform the selected micro-orders.

ROS Decoders

The ROS decoders use the bits from the ROS word to develop control lines. One micro-instruction may activate a number of control lines. Timing consideration governs the source of the lines, i.e., sense latches, ROSDR, ROSDR latches.

Field A (bits 6–9 of the ROS word) in Figure 2-10 is an example of a decoding network to develop control lines from ROS bits. Line A ['gate M1M2 to PAL(64,65)'] is decoded from ROSDR because it updates PAL(64,65)

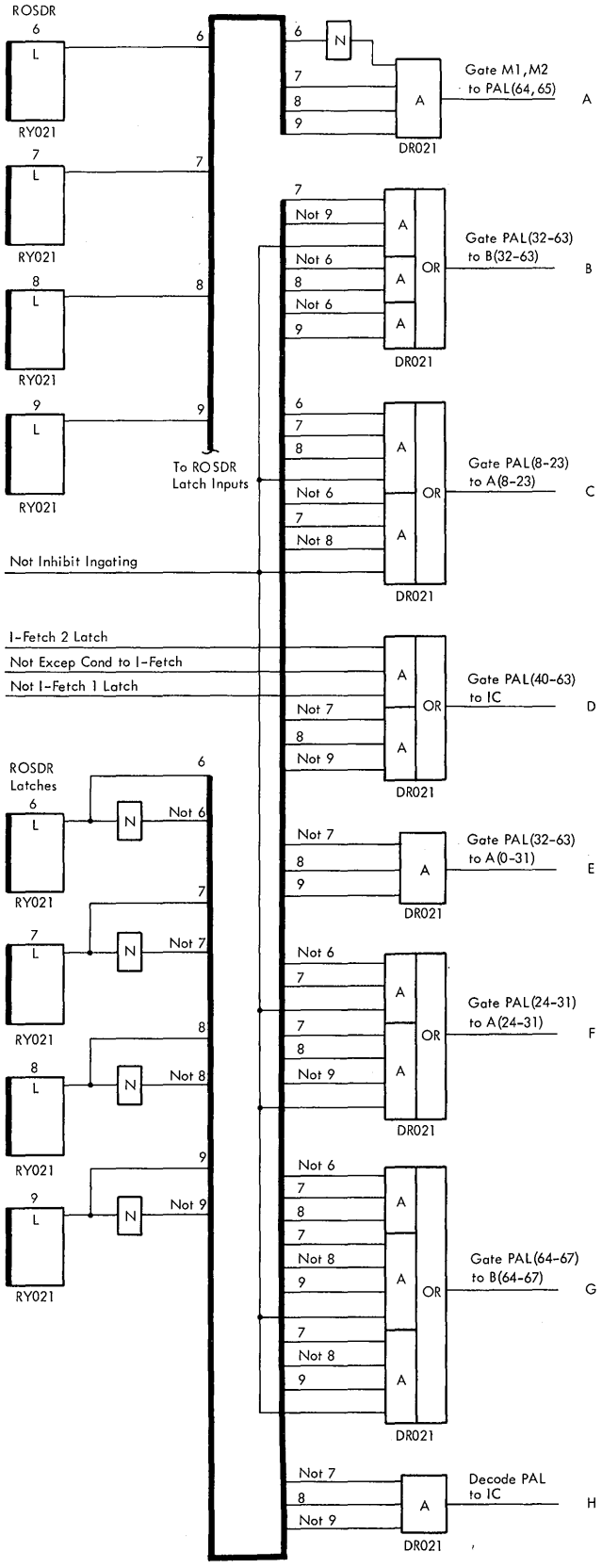


Figure 2-10. Control Field A Decoder

before PAL is gated to AB. The 'B38M' micro-order (A7), shown on ALD M7001, uses line A to update PAL(64,65) during not-clock time. Then, lines B, F, and G are developed from ROSDR latches 6–9 to gate PAL(24–67) to AB(24–67) during the next clock time. Note that the 'B38M' micro-order results when bit 6 = 0 and bits 7, 8, and 9 = 111.

This example demonstrates the register-to-latch-to-register timing that controls the source of the decoded control lines. The other micro-order control fields are decoded in a similar manner to provide the control lines at the proper time.

ROS Timing

ROS timing is controlled by the master clock signals. At P0 + 160 ns, the ROS word is strobed (gated) into the sense latches, which are reset at P0 + 120 ns. Data from the sense latches is stable and available at P0-5 ns when it conditions ROSDR(6–42) for setting at clock P0 and ROSDR(69–84) for setting at clock P2. Gate controls from the sense latches (register data transfer) are activated at clock P2, and remain up for 190 ns. ROSDR latches are set at P7 ('not clock P0-1' signal) and initiate register inputs during the following 200 ns. Figure 2-11 shows the timing relationships of the registers and latches. These timings are theoretical and do not show the delays caused by the signals passing through inverters.

Note: Initially, ROSDR is set to all 1's; a 0 in a sense latch position resets the corresponding ROSDR position.

Maintenance Aids

When an error occurs, data leaves ROS registers and latches before the clock is stopped. To retain this information, which identifies the instruction that resulted in the error, secondary registers (which have no other purpose) are provided: ROSAR latches, ROS previous address registers A and B, and ROS backup register.

ROSAR Latches

The ROSAR latches are loaded from ROSAR at P11 time (not-clock P3 time) of each ROS cycle. At P4 time, the latch output is gated to the previous ROS address registers A and B (PROSAR A and PROSAR B) by an alternator. At the next P10 time (not-clock P2 time), the ROSAR latches are reset.

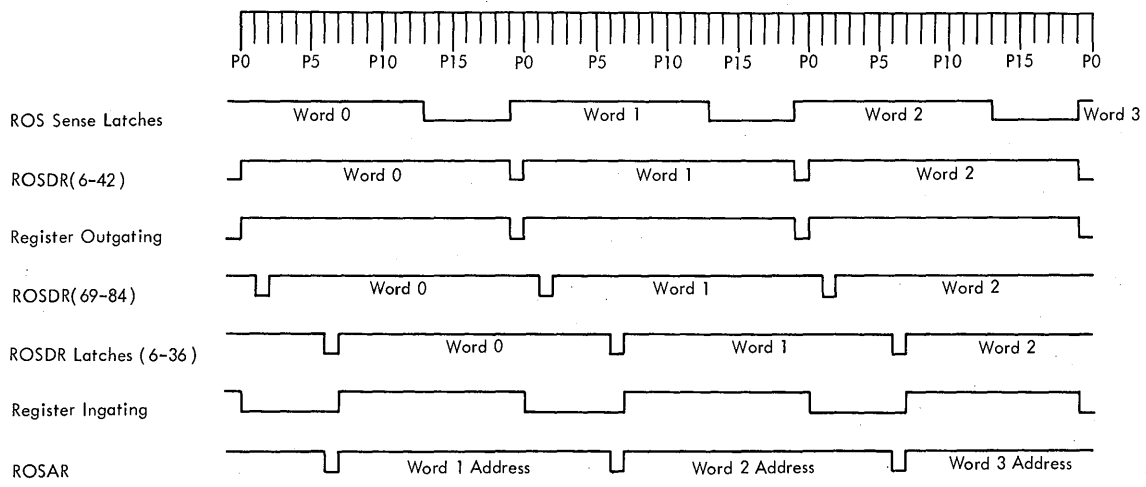


Figure 2-11. Detailed ROS Timing

Previous ROS Address Registers

The contents of the ROSAR latches are alternately gated to PROSAR A and PROSAR B, which alternately contain the address of the current and previous ROS words. These registers, which are loaded at P4 time of the ROS cycle, comprise polarity-hold circuits and retain their values until gated into again. Thus, if PROSAR A is loaded on one cycle and PROSAR B on the next cycle, the contents of PROSAR A are maintained until the third cycle, at which time a new address is loaded. The contents of PROSAR A and PROSAR B are indicated on the roller switch indicators: roller 1, position 4, bits 12–23 and bits 24–35, respectively (Diagram 6-2, FEMDM).

PROSAR A and PROSAR B Alternator

The PROSAR A and PROSAR B alternator (Figure 2-12) causes the contents of the ROSAR latches to be sent alternately to the PROSAR A and PROSAR B indicators. Referring to Figure 2-12, assume that the latch is reset, the CE is at clock P0 time, the A-gate is conditioned, and the B-gate is deconditioned.

At the following P3 time, AND 1 becomes conditioned, which in turn conditions AND 3. The output of AND 1 is also sent to AND 2, but AND 2 cannot be conditioned because the A-gate is set. The output of AND 3 gates the contents of the ROSAR latches to the PROSAR B indicator circuits.

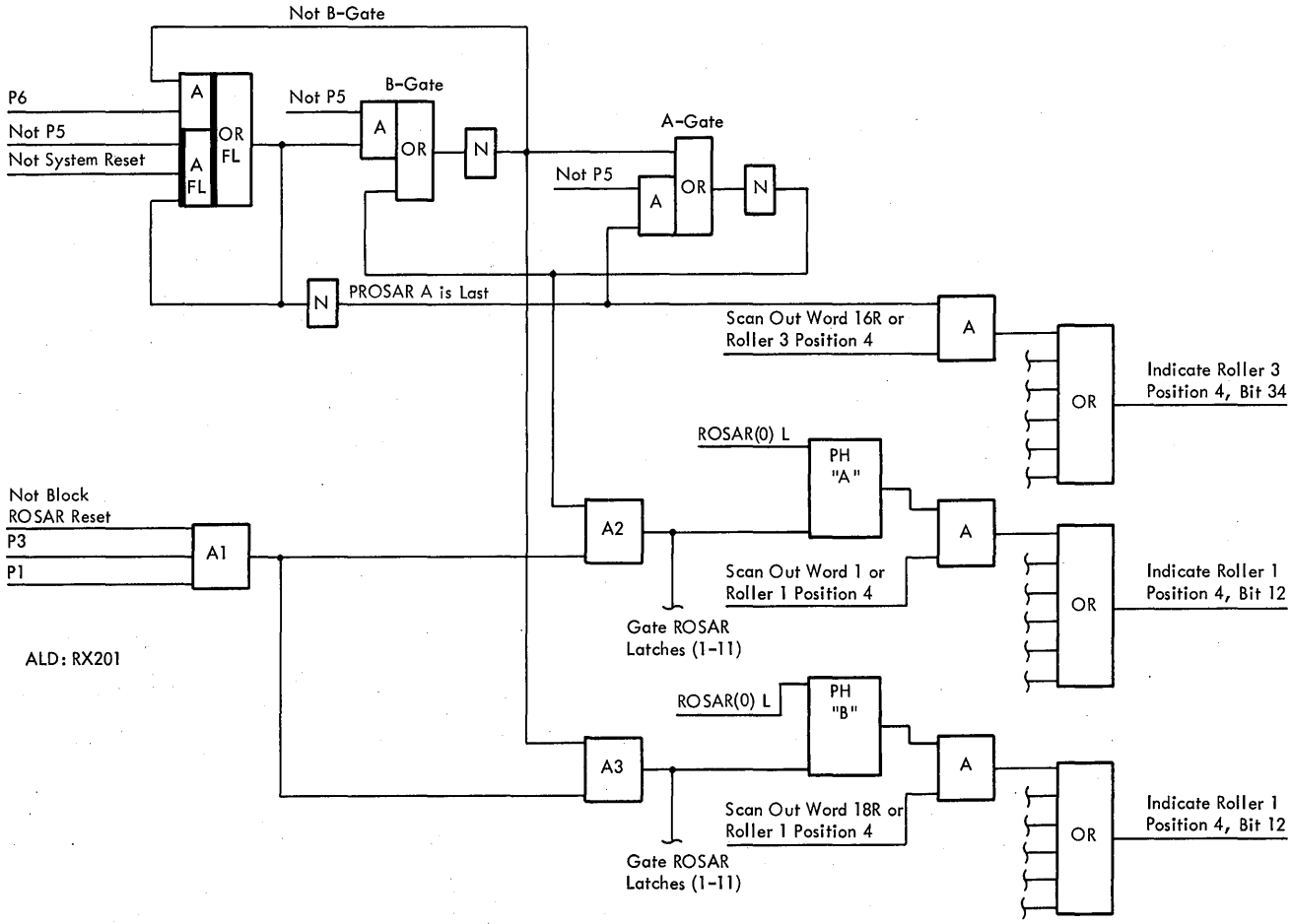
Because the B-gate is deconditioned at P6 time, the latch is set. On the rise of the -P5 signal, the latch being set causes the B-gate to be conditioned and the A-gate to be deconditioned. With the A-gate deconditioned, AND 2 is activated at P3 time (via AND 1) of the following cycle. This action gates the contents of the ROSAR latches to the PROSAR A indicator circuits.

When the -P5 signal drops, the latch is reset. The gates remain in this condition (A deconditioned, B conditioned) until the rise of the -P5 signal. At that time, the A-gate is conditioned by the reset latch, and the B-gate is deconditioned by the A-gate.

During the next two cycles and each cycle thereafter, the operation described above is repeated until the CE clock is stopped. At that time, the contents of the ROSAR latches are gated to the PROSAR indicators associated with the deconditioned gate. To indicate which ROS address is in which set of indicators, the latch output (inverted) is sent to the PROSAR A indicator (roller 3, position 4, bit 34). When the latch is reset, the indicator is on, indicating that PROSAR A contains the address of the current ROS word and that PROSAR B contains the address of the previous ROS word; if off, the contents of PROSAR A and PROSAR B are reversed.

ROS Back-Up Register

The ROS back-up register (ROSBR) holds fields L, NA, K, and J (which are indicated on roller 3, position 4, bits 7–32), and fields R, T, U, and V (which are indicated on roller 4, position 4, bits 17–30). These indicators combined with the ROSDR indicators provide maintenance personnel with a picture of ROS word contents when the CE (with the CHECK CONTROL switch in the STOP position) stops as the result of an error.



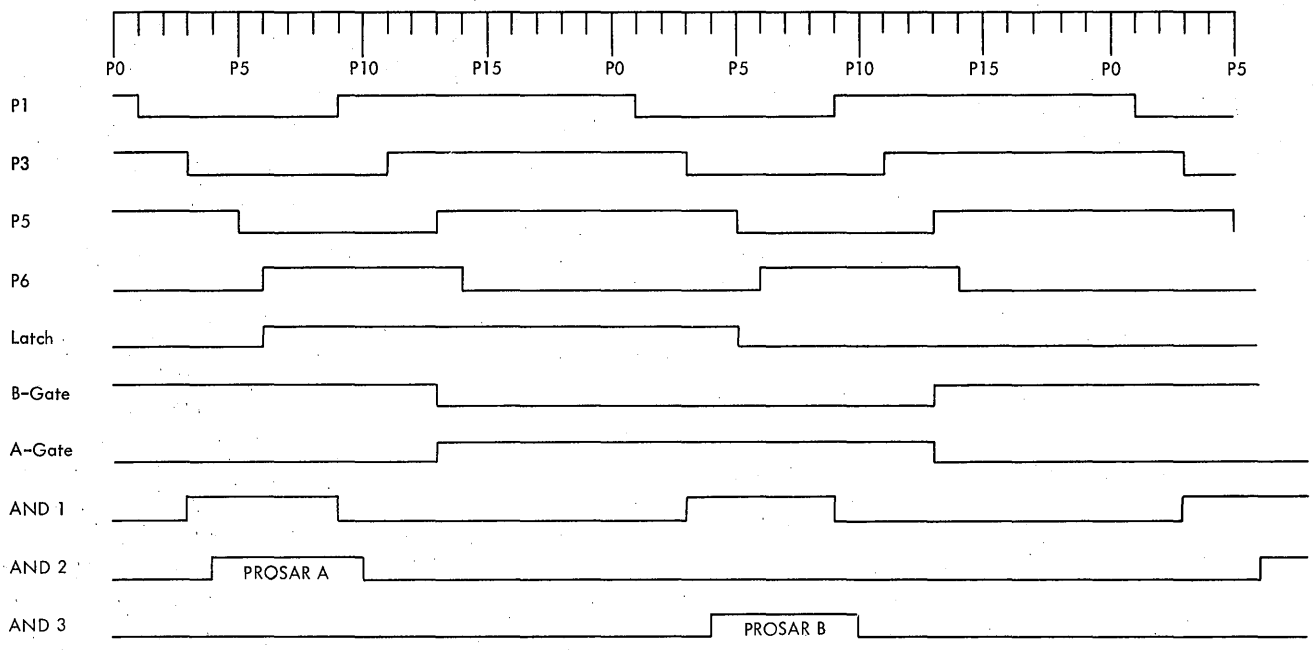
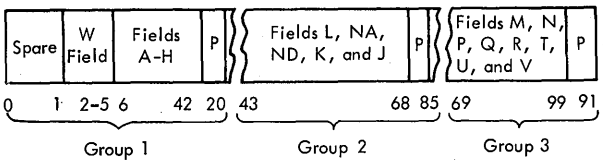


Figure 2-12. PROSAR A and PROSAR B Alternator

ROS Error Checking

Before each ROS word is decoded, it is checked for correct parity. Parity is checked in three groups from the ROSDR and the ROSBR:



Each group contains its own parity bit and must have an odd number of bits to result in correct parity. (There is also a parity bit for the entire ROS word, bit 0. This bit is not checked.) Figure 2-13 illustrates ROS parity checking for the three groups: A of the figure shows how parity is checked from the ROSDR for group 1; B of the figure shows how parity is checked from the ROSBR for group 2; C of the figure shows how parity is checked from the ROSDR and ROSBR for group 3.

The clock reset is blocked in that part of the ROSDR or ROSBR containing the failing ROS word. The part, or parts, not in error are reset, and the next ROS word is gated to its respective register part(s). For example, if bits 43–68 of a ROS word contain an error, the bits are retained for observation. The other two groups not in error (bits 0–42 and 69–99) will change. The groups that change belong to the data word accessed by the failing word when the data register is examined. Thus, when a ROS parity error occurs in one part, the ROS bit indicators on the system control panel comprise bits from two different ROS words.

A ROS parity error also prevents stepping ROSAR, PROSAR A, and PROSAR B, thus enabling the operator to establish the address of the current ROS word, the address of the previous ROS word, and the address of the next ROS word. The address of the previous ROS word should be particularly helpful when parity errors are caused by late ROS branches.

Assume that ROS bit 40 fails. A parity error in bits 2–42 is indicated, and the ROS previous address register indicated the failing word. Reference to the ROS bit plane description shows the expected bit content of the failing word. The incorrect bit (bit 40) can be determined directly by comparing the bit plane description with the indicators.

To summarize, if a machine check is not disabled and a ROS parity error occurs, the parity group in error is not reset at CE clock time of the next cycle. The CE clock set-reset signal is blocked to the group that contains the parity error (Diagram 4-107, FEMDM). The new ROS word, however, is gated to the two groups not in error.

If CHECK CONTROL is in the STOP position, the bit in error can be determined by displaying the ROS micro-instruction, noting which group of bits is in error, deciding which ROS address is in error, and referring to the listing of ROS micro-instructions. If a ROS parity error occurs and the machine check mask bit [PSW(13)] is set, a logout occurs (CHECK CONTROL in PROC).

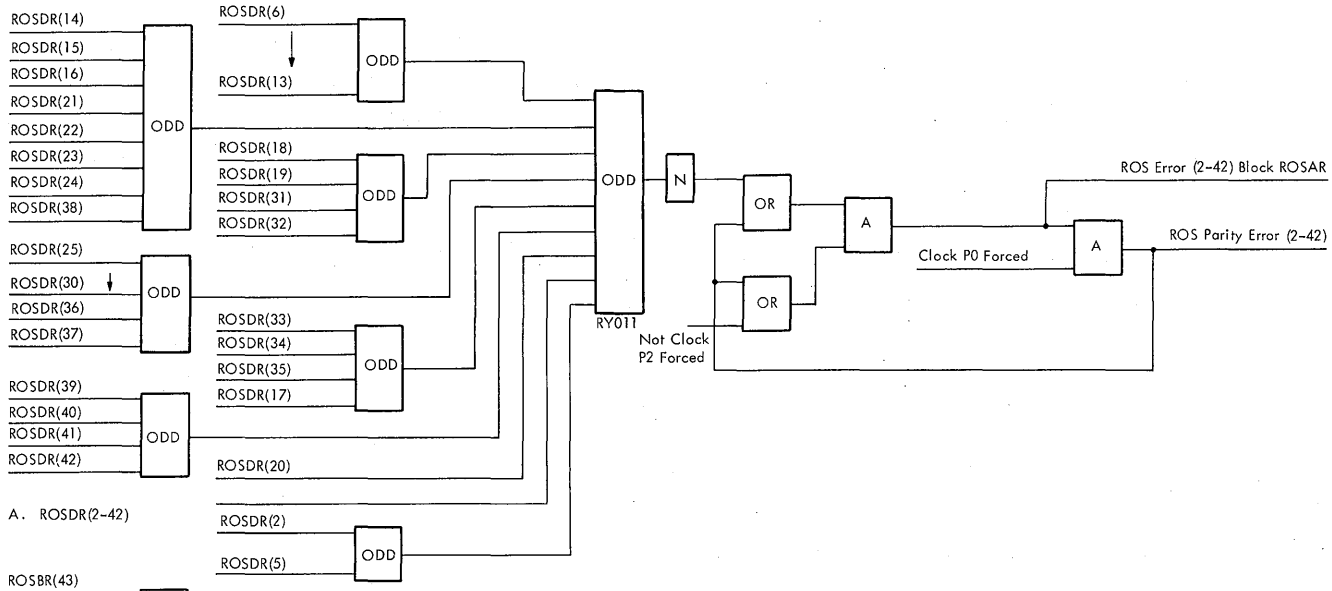
Scan Mode Operations

Scan mode operations affect three fields of ROSDR: field D (bits 17–19), field F (bits 25–30), and field G (bits 31–35). These fields serve dual functions. In the normal mode, they are decoded from the ROSDR latches as standard CE control lines. In Scan mode, they are decoded as special scan control lines and are referred to as field S.

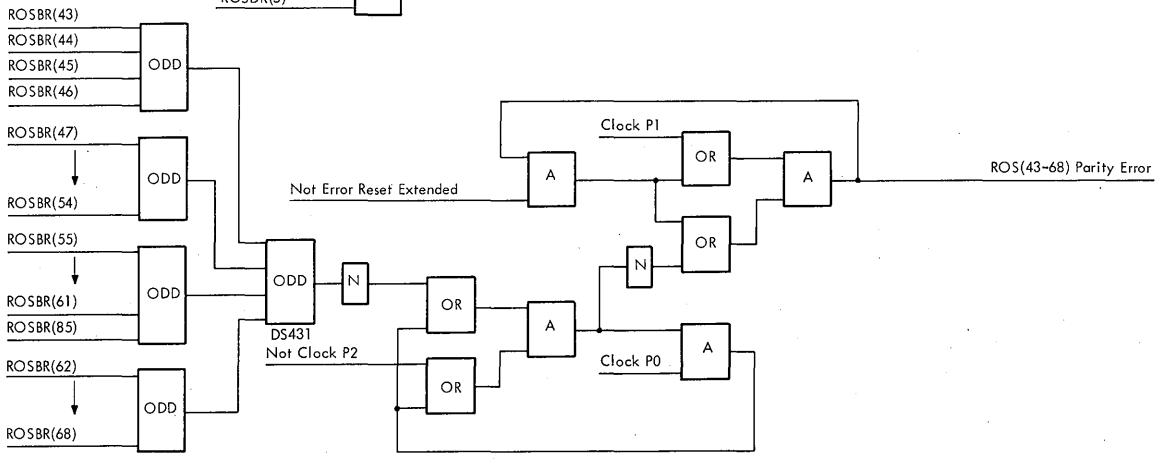
The Scan mode is controlled by the 'scan mode' trigger. When the 'scan mode' trigger is reset, the standard decode path is used. When the 'scan mode' trigger is set, however, the standard control lines are blocked, and scan control lines (using common CE control line codes) are activated.

The 'scan mode' trigger can only be set in Manual mode and reset only in Scan mode. The scan control logic generates an 'inhibit register ingating' signal, which is sent to the ROSDR fields to block register inputs and to allow scan control use of the ROS in sequencing through its test operations.

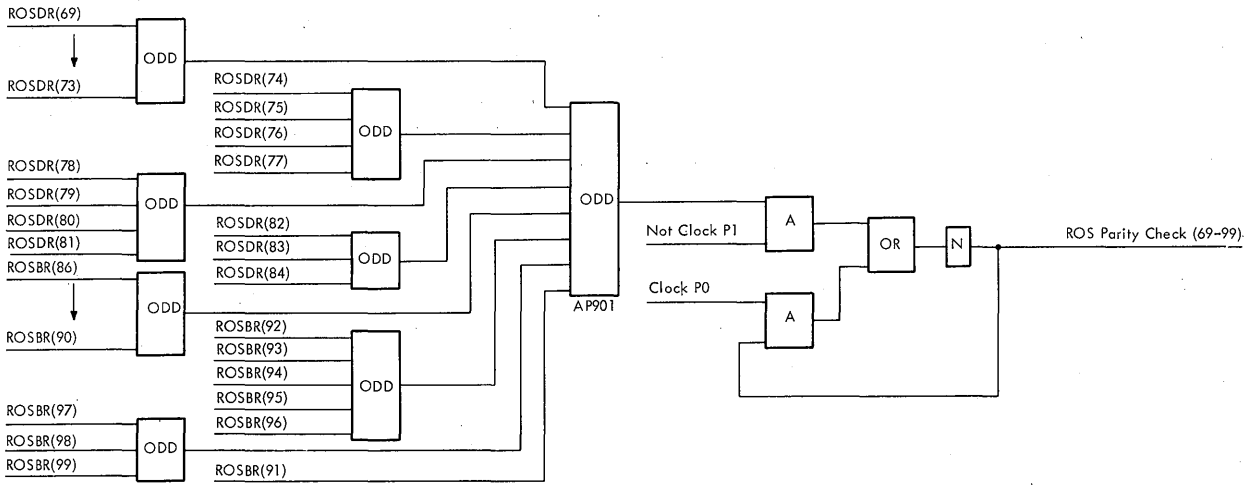
Scan also affects ROS microbranching. (See "Scan Mode Control of ROS" in Section 2 of Chapter 4.)



A. ROSDR(2-42)



B. ROSBR(43-68,85)



C. ROSDR(69-84) AND ROSBR(86-99)

Figure 2-13. ROS Parity Checking

This section describes the registers employed for CE data flow and control functions. For the overall register data flow, see Diagram 2-1, FEMDM.

Q-REGISTER

The Q-register is a doubleword (64 data bits plus 8 parity bits) buffer for instructions entering the CE from main storage on the SDBO (Figure 2-14). Data is transferred from the Q-register to the read or write local storage address latches (LAL), the parallel adder, or the R-register. The contents of the Q-register are displayed at the CE control panel (rollers 3 and 4, position 2).

Input

Instructions are transferred from the SDBO into Q by means of a gate signal decoded from the ROSDR latches.

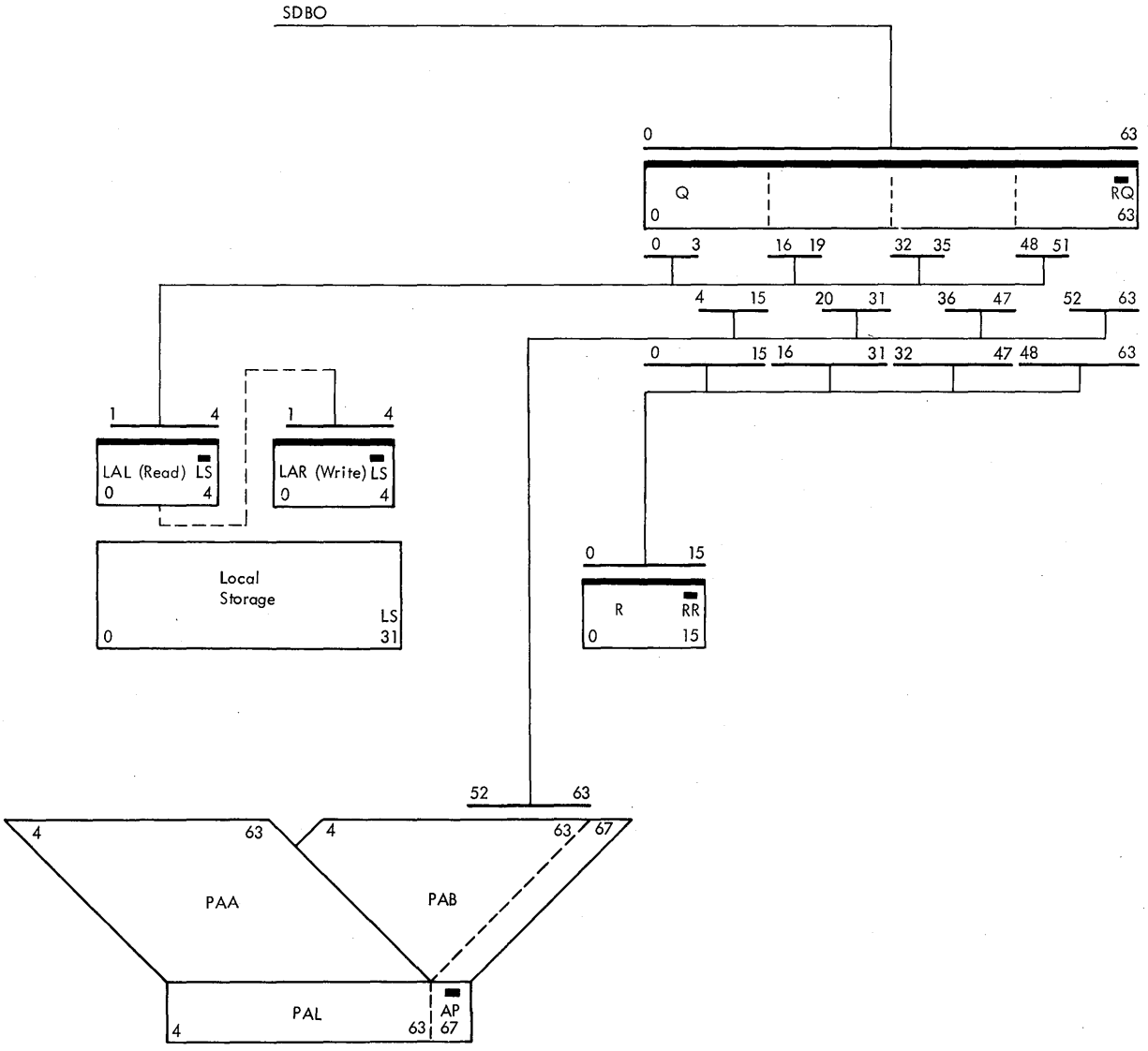


Figure 2-14. Q-Register Data Flow

The transfer of data is initiated by either the I-Fetch hardware or by the ROSDR latches at not-clock time; the transfer controls remain active for one cycle (200 ns). The instructions are transferred into Q at clock time.

Op-Code Transfer

- Only halfwords containing op codes are transferred to R.
- Selection of halfword containing op code is determined by IC(21,22).

Only those halfwords in Q containing op codes are transferred to R. Because RX, RS, and SI instructions are composed of two halfwords and SS instructions are composed of three halfwords (only the first of which contains the op code), it is necessary to select the proper halfword to be transferred to R. Note that because RR instructions are composed of only one halfword, the next halfword to be transferred to R after an RR instruction is completed is the next sequential halfword in Q.

Selection of the halfword for transfer to R is determined by IC(21,22), and transfer is performed either directly through hardware or as a result of an I-Fetch micro-order. Recall that during an I-Fetch operation the op code of the next instruction to be executed is transferred from R to E, with R then being refilled with the next sequential op code. Because the op code of the next instruction to be executed is always in R, its format (positions 0 and 1) can be predecoded to determine the number of halfwords that compose that instruction and thus indicate which of the four Q-register halfwords contains the next sequential instruction op code. This predecoding occurs at end-op time of each instruction; the result (Q halfword number) is set into IC(21,22), which in turn selects a subsequent I-Fetch ROS word that specifies the next op-code halfword to be transferred to R.

Note: IC(21,22) is not used in addressing main storage; it only specifies which of the four Q-register halfwords is to be transferred to R by the following I-Fetch sequence. The IC(21,22) values associated with each Q-register halfword are illustrated in Figure 2-15.

An exception to the normal I-Fetch ROS word method of transferring the Q-resister halfwords to R is as follows. Assume a condition whereby a four-byte (RX, RS, or SI format) instruction occupies the right half of Q, IC(21,22) = 10, and a storage request is generated to main storage. When the I-Fetch sequence loads the op code of this four-byte instruction into R, the predecode logic determines that the next doubleword being accessed from main storage contains, in its leftmost byte, the op code of the next sequential instruction. [IC(21,22) is also stepped to 00 during this particular I-Fetch.] When, during the following I-Fetch sequence, the contents of R are transferred to E for execution, R is not refilled from Q in the normal manner because the particular I-Fetch ROS word selected to control this operation does not contain a micro-order specifying refilling of R. When that doubleword being brought from main storage enters Q, however, Q(0–15) (the op-code halfword) is allowed to proceed directly on into R. Thus, R again contains the op code of the next instruction to be executed, even though the instruction was not present during the I-Fetch sequence. This function is accomplished solely through the use of hardware that constantly tests for the presence of two signals: 'I-Fetch latch 1 and 3 set' and 'IC(21,22) = 00'.

B-Field and D-Field Transfer

The instruction B-field, which specifies LS registers, and the D-field, which is the main storage address displacement, is transferred from Q to LAL and the parallel adder, respectively. To save time, this information is transferred directly from Q instead of from R or E, thus allowing LS and the address to be available before the execution time of the associated instruction. Transferring these fields must be performed selectively so that the information is associated with the correct instruction.

B-Field Transfer

The four-bit Q-fields (B-field address data) are normally transferred to LAL at end-op time, under hardware control, per IC(21,22) or, for certain branch instructions, per D(21,22). (See Diagram 4-201, FEMDM.)

For SS instructions, however, two B-field values must be transferred to LAL. The first B-field is transferred to LAL at end-op time, per IC(21,22), under hardware control and

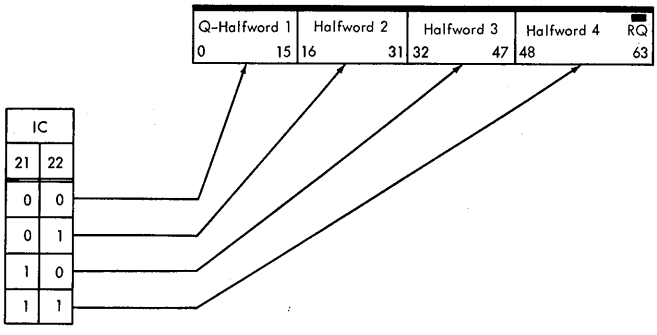


Figure 2-15. Q-Register Halfword Transfer Per IC(21,22)

in the normal manner. [D(21,22) is used for branch instructions.] The B-field of the second operand is then transferred to LAL (from the selected portion of Q) during I-Fetch of the SS instruction by a micro-order contained within one of the I-Fetch ROS words.

All transfer of data from Q to LAL takes place at not-clock time; the data is transferred into LS at clock time.

Note: Because an RR instruction can be contained within R and E, only halfword transfers from Q to R are required for RR instructions. All addresses for LAL can therefore be transferred directly from R or E.

D-Field Transfer

Selection of the Q-register D-field for transfer to the parallel adder (for use in address computation) is determined by the particular ROS word selected for use. The D-field transfer occurs at clock time.

R-REGISTER

The R-register is a halfword (16 data bits plus 2 parity bits) register that provides intermediate buffering between Q and E for the halfword that contains the op code (Figure 2-16). This intermediate buffering speeds the refill of Q by allowing a storage request when the last op code has been transferred from Q but has not yet been executed. The R-register is displayed on roller 5, position 3, bits 0–17.

Input

The R-register is loaded with one of the four halfwords in Q at I-Fetch time under ROS control. The contents of PAL(56–63) are also transferred to R (at I-Fetch time of the subject instruction of an Execute instruction).

Output

Whenever the instruction in R is predecoded as an RR nonbranch instruction, R(8–11) is transferred to LAL at end-op time (Diagram 4-202, FEMDM). (The RR format indicates that R contains the entire instruction.)

Whenever the instruction in R is predecoded as an RR branch instruction, R(12–15) is transferred to LAL at end-op time. The contents of R are transferred to E at I-Fetch time under ROS control.

Predecoding

The R-register predecode logic samples R(0,1) at end-op time to determine the format of the next instruction. Time is saved because prefetching of operands per the format prepares data for use after the instruction is transferred to E. In addition, R(0,1) and IC(21,22) determine the need for storage requests to refill Q.

R(0–4) is tested for shift instructions. Because shifting does not require a storage request, time is saved if a shift instruction is decoded when a Q-refill request is generated 2 cycles before end op. The Q-register refill exceptional condition is eliminated because there is no interference between the shift instruction and the Q-refill storage request.

R(0–7) is sampled for branch instructions so that prefetching of the new instruction address can start immediately, thus saving time. R(12–15) is sampled for a zero condition, which prevents the branch in the RR format.

On a branch end op, the instruction halfword is still in the process of being requested from storage. To save time in prefetching operands, the instruction format is predecoded from the SDBO rather than waiting until the instruction becomes stable in R.

E-REGISTER

The E-register is a halfword (16 data bits plus 2 parity bits) register which contains the op-code halfword of the instruction being executed (Figure 2-17). The E-register contents are displayed at the CE control panel (roller 5, position 3, bits 18–35).

Input

An op-code halfword (including two parity bits) is transferred from R to E during a normal I-fetch sequence under ROS control. On shift operations, D(18–21) is transferred to E(12–15) via the E-register incrementer. The data path from PAL(56–63) to E(8–15) is used in some SS format instructions to control the specified number of bytes.

Output

Op-code signals to control processing are decoded directly from E(0–7) without the use of gating logic.

LS is addressed by transferring E(8–11) or E(12–15) to LAL(1–4). E(8,11,12,15) is examined for an LS address

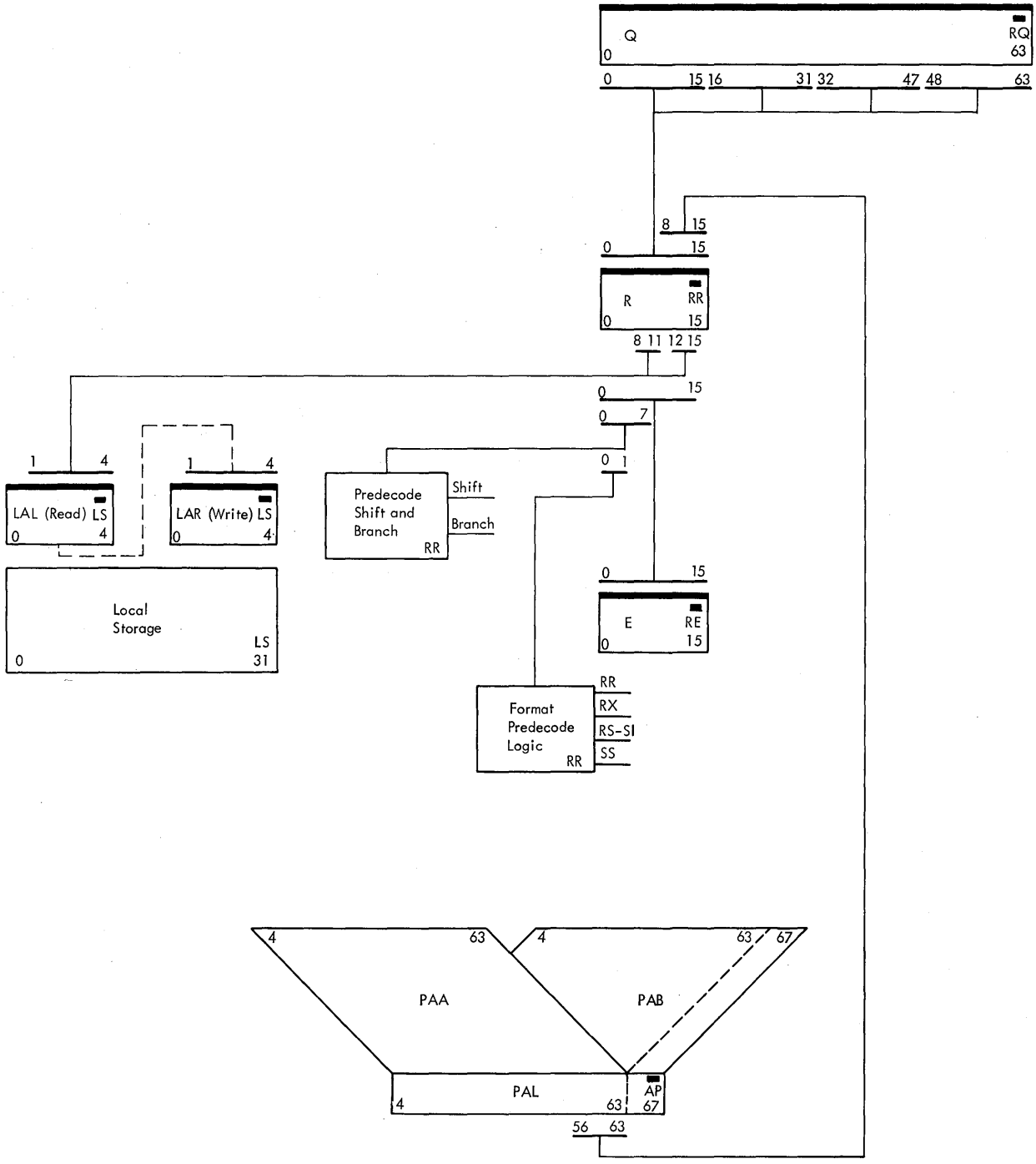


Figure 2-16. R-Register Data Flow

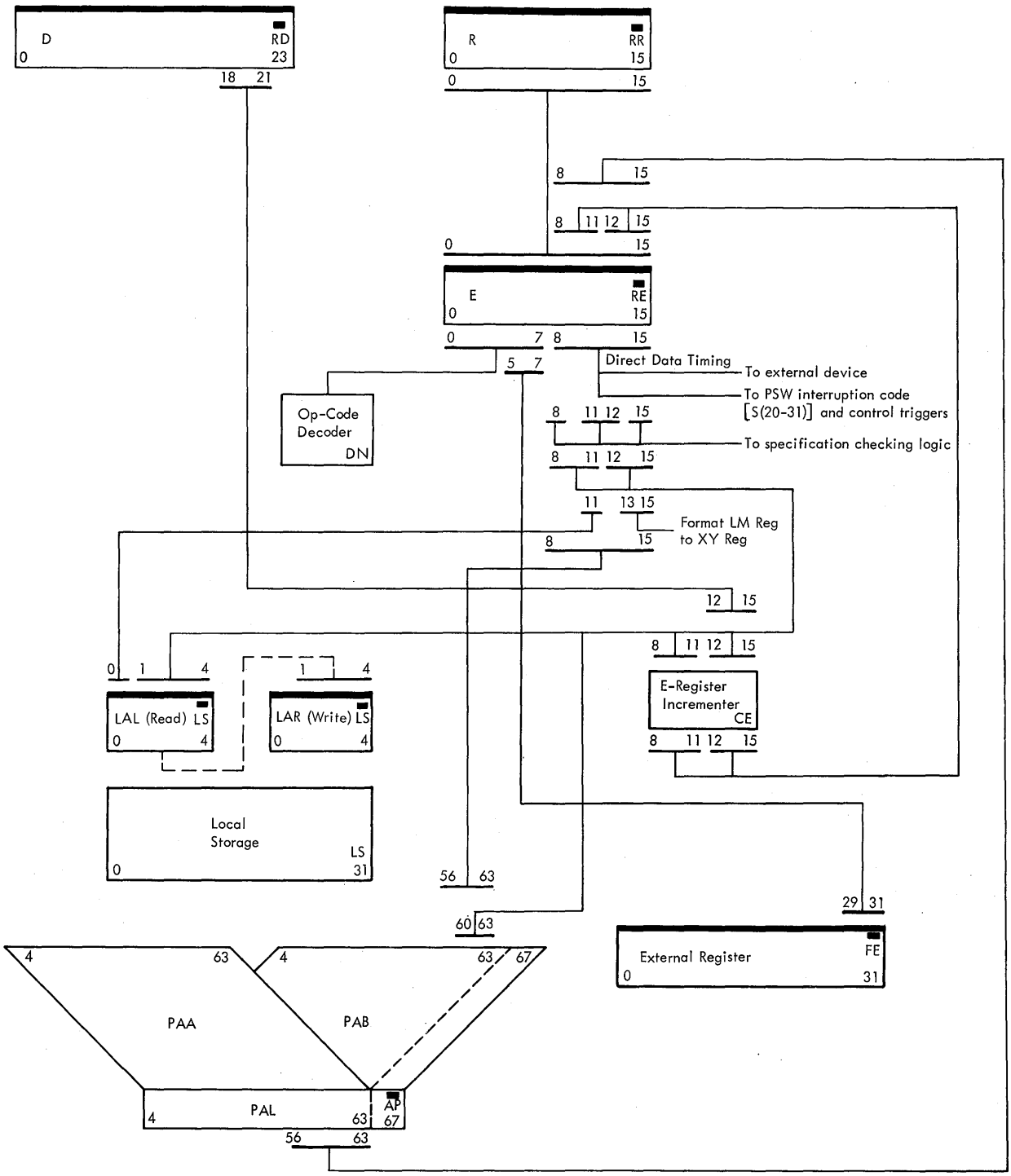


Figure 2-17. E-Register Data Flow

specification error. When manual operations are performed using LS, E(11,12–15) is transferred to LAL(0,1–4) so that all registers may be addressed. Transferring E(8–11) to PAB(56–59), E(12–15) to PAB(60–63), and E(8–15) to PAB(56–63) provides for multiply and divide operand aligning, byte count control for SS format operation, and subsequent transfer to other registers. E(8–11), E(12–15), or E(8–15) may be sent to the E-register incrementer for alteration under ROS control.

For Direct Control instructions (WRD,RDD), E-Reg bits (8–15) are decoded to select the receiving CE or IOCE and to develop the 'simplex' signal to inform the element of the operation it is to perform. These are gated by the 'timing gate' trigger. E(8–15) is tested for a specification error in an SIOP operation. E(8–15) is also sent to the PSW interruption code [S(20–31)] on a supervisor call interruption only, and to control triggers such as 'disable interleaving' and 'diagnose FLT'. During execution of display instructions, E(13–15), under ROS control, are used to format the contents of the LM-registers into the XY-registers.

Incrementers

Two four-position incrementer registers are available, with ROS controls, for either treating E(8–11) and E(12–15) separately or treating E(8–15) as an entity. Positions E(8–11) and E(12–15) can be either incremented or decremented by 1, but E(8–15) can only be decremented by 1 (for example, used for reducing length fields in logical VFL operations).

The E-register incrementers consist of latch circuits with logic decoder inputs (Diagram 4-203, FEMDM). The four-position incrementers are not capable of counting; rather they decode the binary information at their input, generate a binary value of 1 greater (or 1 less), and then set that value into the latches. Processing E(8–15) as an entity is accomplished by logically connecting the two four-position incrementers.

ROS controls also load constants into the E-register incrementers during execution of certain instructions (e.g., fixed-point multiply or divide) in order to select serial adder positions when developing products or quotients.

During shift instructions, D(18–21) is gated into the E(12–15) incrementer; decrementing functions reduce the specified shift amount as each shift operation is completed and, thus, control the shift instruction.

When E(8–15) is modified in the E-register incrementer, E(8–15) parity may change. Diagram 4-204, FEMDM, shows the parity prediction logic to yield correct parity for E. If, for example, E(15) = 0 and the 'add 1 to E(12–15)' signal is active, the 'change parity of E(12–15)' signal is developed. Assuming E(8–15) is odd, the 'INCR(8–15) bits even' latch is set, which, in turn, sets the 'E(8–15) parity' trigger. Thus, parity is altered at the same time E is modified.

INSTRUCTION COUNTER

The instruction counter (IC) is a 24-bit (plus three parity bits) register used primarily in addressing doublewords of instructions from main storage (Figure 2-18). IC bits 0–23 are displayed at the CE control panel as IC bits 8–31 (roller 6, position 3, bits 9-35).

Input

PAL(40–63) is transferred to IC(0–23) when incrementing the IC, or when entering a branch instruction as specified by D. Because IC(21,22) selects halfwords in Q, ROS controls the setting of IC(21,22) independently of the parallel adder.

During execution of the 9020E Display instructions, IC(0–20) can be loaded, under hardware control, from SDBO(40–60). In this case, the SDBO(40–60) specifies the address of a page (512 bytes) in main storage.

Output

IC(0–23) is transferred to PAB(40–63) to be incremented by 8 so that the next sequential instruction address in main storage will be specified by the IC. When called for, the output of the IC is gated to the logical address bus (LAB), bits 0–11. If LAB(0–11) is equal to 0, physical PSBAR(9–12) and logical PSBAR(13–19) are gated to SAB(1–11). If LAB(0–11) is not equal to 0, LAB(1–4) is used to select an ATR slot which contains the physical address of the storage element (SE/DE) to be selected. The physical address is gated to SAB(1–4), and IC(5–11) is gated to SAB(5–11). SAB(12–23) is gated directly from IC. IC(23) is transferred to the specification logic to test for a 0-bit on instruction addressing; a 1-bit indicates a specification error.

In some instances, the address is for VFL data. Accordingly, IC(21–23) is transferred to ABC(0–2) on VFL operations to specify the desired data byte in AB. IC(21,22), through ROS branching, specifies the Q-halfword to be transferred to R; IC(23) determines the byte within that halfword.

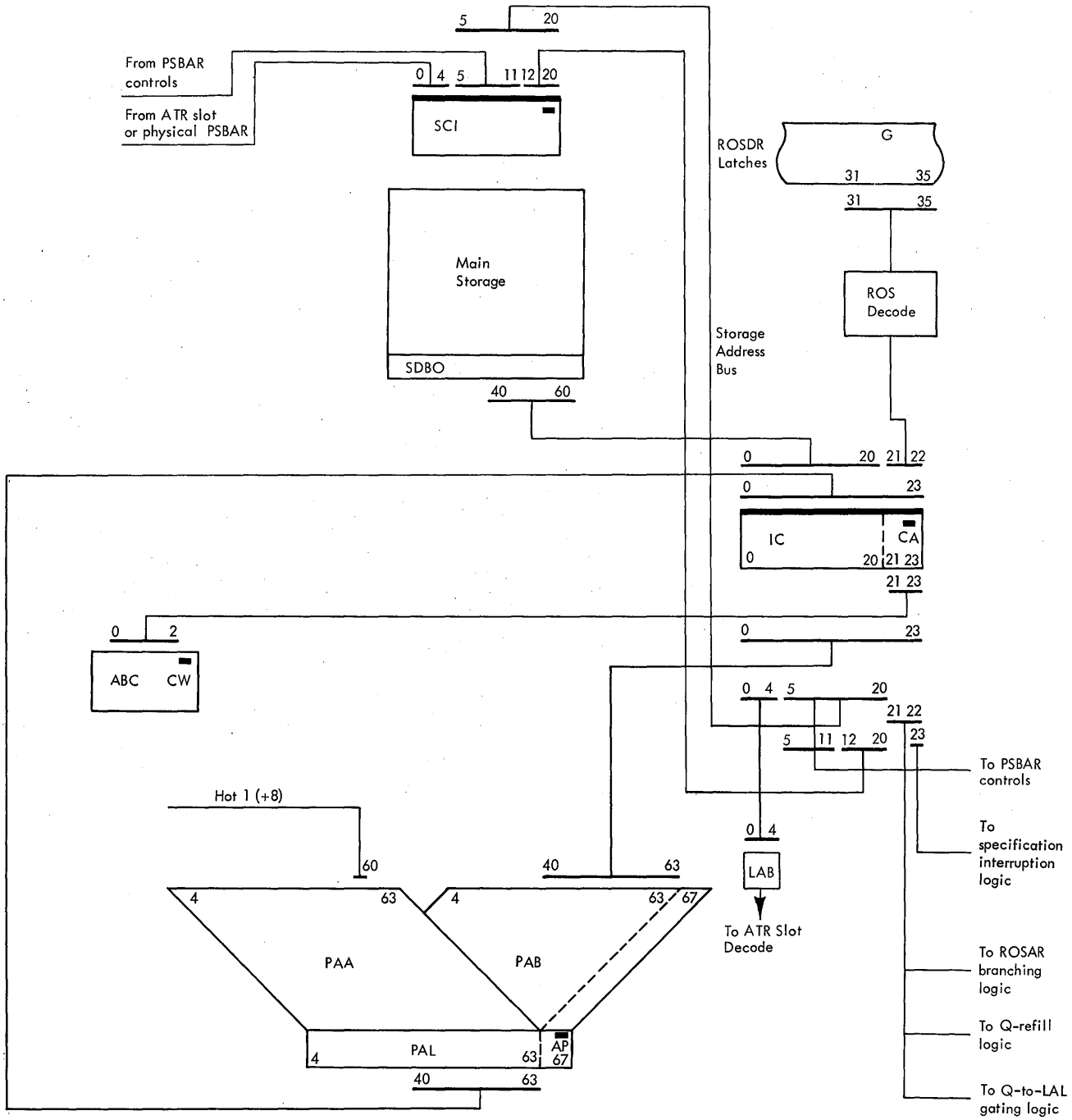


Figure 2-18. Instruction Counter Data Flow

Incrementing IC(0–20)

- After each storage request is generated, IC(0–20) is incremented by 8 to develop address of next sequential doubleword in main storage.

Incrementing (updating) IC(0–20) by 8 to develop the address of the next sequential doubleword in main storage is accomplished using the parallel adder. [IC(0–23) is gated to PAB, and a hot-1 bit is forced into PAA(60).]

At any given time, however, IC(0–20) may be either one or two doubleword addresses ahead of the doubleword in which the instruction being executed is located. These conditions occur as follows. When the instruction being executed is contained in a doubleword still present in Q, IC(0–20) has been updated and is one doubleword (8 byte addresses) ahead of the doubleword in Q. However, when the op-code halfword just transferred to R happens to occupy the last halfword portion of Q, [IC(0–20) already being one doubleword address ahead], a storage request is generated to access the next doubleword and the IC is again updated by 8. [IC(0–20) is now two doubleword addresses ahead of the doubleword in which the instruction being executed is located.]

Incrementing IC(21–23)

- After each op-code halfword is transferred from Q to R, IC(21,22) is set to the value corresponding to the Q portion occupied by that halfword.

IC(21,22) values of 00, 01, 10, and 11 correspond respectively to the four (1–4) Q-register halfword portions. On the same cycle in which the op-code halfword is transferred from Q, IC(21,22) is set to the value corresponding to that halfword portion. [IC(21–23) trigger circuitry is not capable of accumulating, but only of receiving, input values. Thus, these triggers are not stepped or incremented but, rather, set to values indicating the four Q-register halfword areas.] IC(21,22) is controlled by ROS words (in the case of instruction sequencing, by the same ROS words that gate the Q-register op-code halfword to R).

In the event of a non-RR instruction, IC(21,22) must be changed by 2 or 3 to skip over the non-op-code halfwords remaining in the instruction. This skipping is accomplished as follows. Two factors involved in the 64-way ROS branch (NEXT-INST*IC) occurring at end-op time of each instruction are: (1) the format of the instruction op code previously transferred to R, indicating the number of halfwords composing that instruction; and (2) the contents of IC(21,22), indicating the Q-register area occupied by that instruction. These factors enable the end-op branch to access the proper I-Fetch ROS word for gating out the next sequential op-code halfword and also setting IC(21,22) to the value corresponding to that particular halfword area.

IC(23), set only during VFL operations in which an odd-numbered operand address is set into the IC, is not otherwise subject to change.

Note: IC(21–23) of the VFL operand addresses is placed into the AB counter, which then assumes the function of sequencing through the data-field bytes.

When IC(21,22) is set to new values, the parity of IC(16–23) may change. Parity adjust logic (Diagram 4-205, FEMDM) conditions the IC P(16–23) bit when the 'set IC(21,22)' micro-order is executed. IC(21,22) is set before the 'adjust parity' trigger is set, but circuit delays hold the parity adjust condition until the trigger is set. In effect, the parity adjust logic subtracts the parity of the old value of IC(21,22) and then adds the parity of the new value of IC(21,22), thus resulting in an updated IC P(16–23).

D-REGISTER

The D-register is a 24-bit (plus three parity bits) register which functions as a main storage address register for certain operations and as an I/O channel and unit address register for I/O instructions (Figure 2-19). D-register bits 0–23 are displayed at the CE control panel as D bits 8–31 (roller 1, position 2, bits 9–35).

Input

Inputs to D are under ROS control. Main storage address information may come from either the parallel adder or the ADDRESS switches. Address information placed into D(17–20) is generated by the interruption logic.

During execution of the 9020E Display instructions, D(0–20) can be loaded, under hardware control, from SDBO(40–60). In this case, the SDBO specifies the address of a page (512 bytes) in main storage.

Output

When called for, the output of D is gated to the logical address bus (LAB) bits 0–11. If LAB(0–11) is equal to 0, physical PSBAR(9–12) and logical PSBAR(13–19) are gated to SAB(1–11). If LAB(0–11) is not equal to 0, LAB(1–4) is used to select an ATR slot which contains the physical address of the storage element (SE/DE) to be selected. The physical address is gated to SAB(1–4), and D(5–11) is gated to SAB(5–11). SAB(12–23) is gated directly from D. D(0–20) is transferred to the SCI to provide storage addressing. D(18–21) is sent to the E-register, via the E-register incrementer. D(21–23) is sent to the ST byte counter, ROSAR branching logic, and

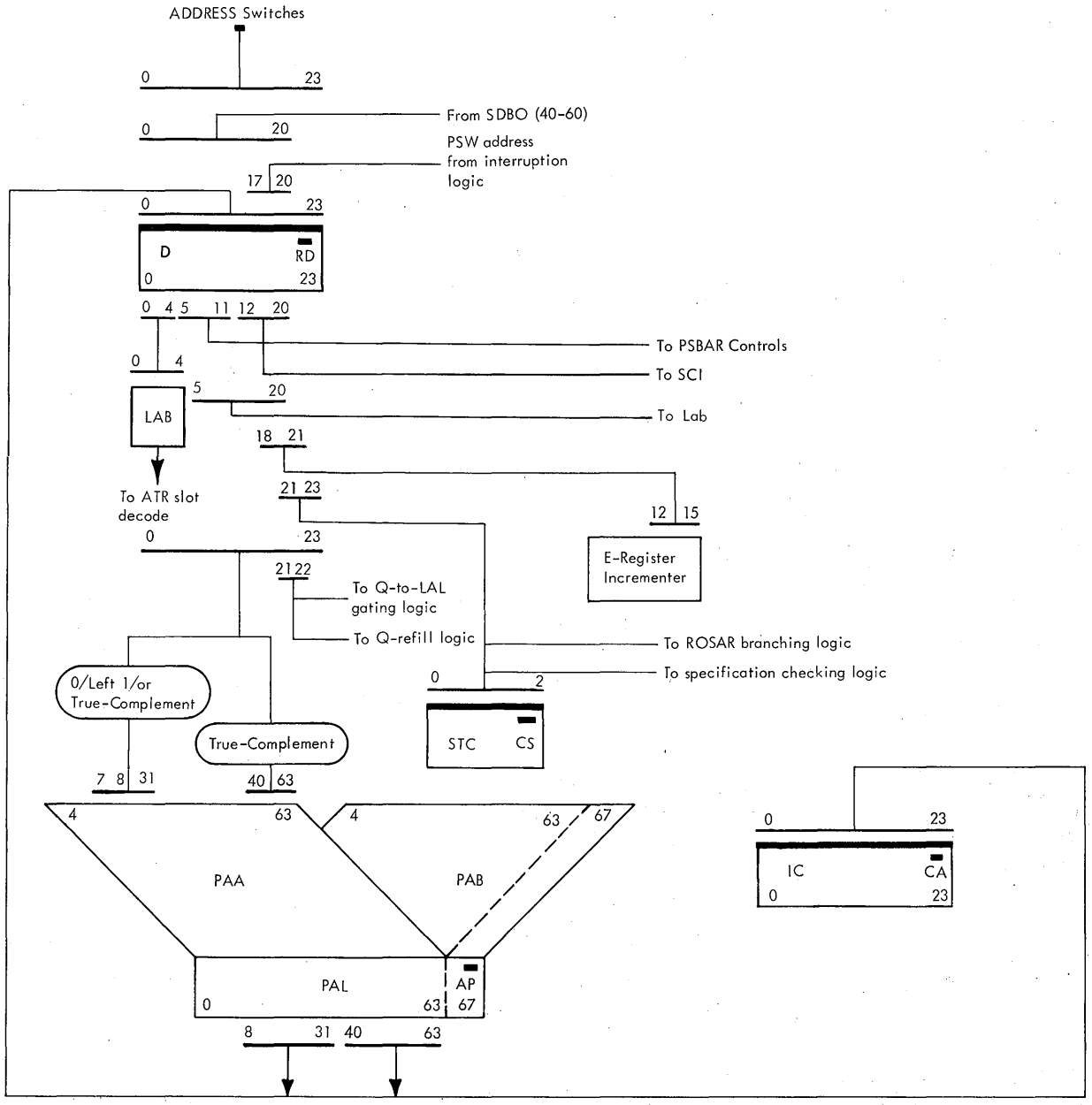


Figure 2-19. D-Register Data Flow

specification checking logic. D(21,22) determines which halfword of Q to use to provide LS address information. The transfer of D(0–23) to PAA(8–31) or PAA(40–63) provides the path to alter or update the D-register.

Operational Functions

Operations in which D participates are: (1) branch and execute, (2) shift, (3) VFL, (4) fixed-point, (5) floating-point, (6) manual-control, and (7) interruption.

Branch and Execute Operations

For branch and execute instructions, the specified successful branch address (all branches are assumed to be successful until otherwise determined) is placed into D by the normal I-Fetch sequence. A storage request is then issued to the SCI per the D-register. D(21,22) specifies the particular op-code halfword within the doubleword in the same manner as IC(21,22) does for normal operation. If, during execution of a branch instruction, the branch is found to be successful (branch condition satisfied), the requested doubleword from main storage is gated into Q, and the branch address in D is sent to the parallel adder and updated by 8. The result is placed into the IC(replacing the IC address), and the program proceeds in the normal manner. If, however, the branch instruction is found to be unsuccessful, the doubleword requested from main storage (per D) is not gated into Q, and the branch address in D does not replace the IC address. The program then proceeds with the next sequential instruction, per IC.

Shift Operations

For RS instructions, I-Fetch adds the base and displacement values and places the result into D. Normally, this result is the second operand address. For shift instructions, however, this total specifies the number of bit positions to be shifted, and is used as follows. The number of shifts specified by D(22,23), a maximum of 3, are executed immediately upon being computed in the parallel adder and without the use of D. The number of shifts remaining is now specified by D(18–21), which indicates the number of shift operations necessary to complete the shift instruction, provided four shifts are accomplished by each shift operation.

Because left 4 and right 4 shifts are possible in the parallel adder, the binary number in D(18–21) is transferred to E(12–15), where the E-incrementer then controls the remaining number of left 4 or right 4 shift operations required to complete the instruction.

VFL Operations

For VFL operations, destination operand addresses are placed into D by the I-Fetch sequence. (Source operand addresses are placed into the IC.) Storage requests for destination operands are made per D(0–20), and the accessed doubleword is loaded into ST. D(21–23) is set into the ST byte counter to control ST byte transfer. The address in D is updated by 8 following each storage request, and, when the ST byte counter value reaches 7, another storage request is made per D to refill ST with destination operand data.

Fixed-Point Operations

For fixed-point operations, operand addresses are placed into D by the I-Fetch sequence. Operand storage requests are made per D(0–20), with D(21) determining which 32-bit word of the accessed doubleword is to be gated into ST. For halfword operation, D(22) determines which half of the 32-bit word specified by D(21) is to be gated into ST.

Floating-Point Operations

For floating-point operations using long operands, D and T provide for the handling of a 56-bit fraction. The high-order 24 bits of long fractions are contained in D.

Manual-Control Operations

In manual-control operations (manual mode), addresses entered into the ADDRESS switches are transferred to D. A storage request is then made per D to reference main storage for operations such as storing and displaying.

The address is entered into D as follows. Manual operation microprogram routines (for example, store or display) cause the parallel adder to generate all 1's and then transfer them to D. Those ADDRESS switches not depressed (not set to 1) cause their associated D-register positions to be reset to 0; the resulting bit configuration in D is the address.

Interruption Operations

At end op of each instruction, ROS Y-branch (overriding branch) tests are made to check for the presence of any interruptions. Each interruption forces a unique bit configuration into D(17–20), which is generated by the interruption-decode and forced-address logic. (This logic also forces an address into ROSAR to access the first ROS

word of the associated interruption-handling routine.) These four positions constitute the low-order bits of doubleword addresses in main storage that contain the new PSW for the various interruptions.

AB REGISTER

The AB register is a doubleword (64 data bits plus 8 parity bits) register that serves as a working register and as a buffer for doubleword operands received from main storage (Figure 2-20). Note that the AB register is logically divided into two 32-bit (plus four parity bits) registers, A and B, and has a four-bit extension, B(64–67), to retain low-order significance during certain shift and arithmetic operations. The contents of AB are displayed at the CE control panel (rollers 3 and 4, position 3).

Input

All AB positions are reset at clock P1 time of the cycle in which they are selected to receive information; data transfer then takes place at P2 time.

Main storage information (doubleword length) is transferred into AB under ROS control by transferring SDBO(0–31) and SDBO(32–63), plus parity, to A and B, respectively.

Parallel adder information (plus assigned parity) is also transferred to A and B under ROS control. Ingating of B(64–67) from PAL(64–67) or PAL(28–31) provides for maintaining high- and low-order significance during shift operations. Transferring PAL(24–31) to A(24–31) facilitates processing of fixed-point divide instructions.

Output

All AB transfer is under ROS control and is accomplished primarily through the use of gate-control triggers. All gate-control triggers are reset at P1 time of each machine cycle; the specific triggers selected for use are set at P2 time of the cycle in which they are to function. (One level of logic delay is incurred in transition; as a result, the respective transfer controls are activated at P3 time of that same cycle.) Selection of the gate-control triggers for use during any given cycle is determined either directly (through ROS decoding) or indirectly (through the AB byte counter). Parity information is transferred on a byte basis.

On multiply operations, the partial product, B(66,67), is placed directly into the serial adder latches (SAL) per E(14,15). Divide operations transfer A(4) or A(28) to one position of SAL per a ROS micro-order and E(14,15).

ST REGISTER

The ST register is a doubleword (64 data bits plus 8 parity bits) register that serves as a buffer between main storage, LS, and the CE and also serves as a working register for arithmetic and logical operations (Figure 2-21). Note that the ST register is logically divided into two 32-bit (plus four parity bits) registers. The contents of ST are displayed at the CE control panel (rollers 1 and 2, position 3).

Input

- Inputs are from main storage, LS, parallel adder, serial adder, PSW register, interrupt logic, DATA switches, and the 9020 out bus.

When new data is transferred into ST, only the bit positions involved are reset. Resetting occurs at P1 time and data transfer at P2 time. Reset signals are generated by the gating signals so that doublewords may be assembled (Diagram 4-206, FEMDM).

Main storage information is placed into ST by transferring SDBO(0–63) to ST(0–63), SDBO(0–31) to S(0–31) or T(32–63), or SDBO(32–63) to T(32–63). LS information (32 data bits plus 4 parity bits) is transferred to either S or T. All ST storage activity is controlled by ROS.

ROS also controls the loading of ST from the following 9020 registers: the select-register, the address translation register (ATR), the diagnose accessible register mask (DAR mask), CCR, PSBAR, G register, external bus, and the external-register. This information is transferred to the ST via the 9020 output bus and the LS output bus.

PAL(32–63) or PAL(40–63), plus parity, is transferred to T, and SAL(0–7), plus parity, is transferred to the ST byte per the ST byte counter and incrementer. ROS controls the transfer from the adders to the ST register.

PSW information is transferred from the PSW register to S(0–15) and T(32–39) under ROS control. The interruption code from the interrupt logic enters S(20–31) and is stored into the old PSW. Figure 2-22 shows S(20–31) input logic.

For manual control operations, information from the DATA switches is placed into ST for subsequent entry into main storage (or LS) in the following manner. All positions of ST are set to 1's by means of the parallel adder and LS ROS micro-orders. All DATA switches not set to 1's cause their respective ST positions to be reset to 0. Thus, ST reflects the information contained in the DATA switches.

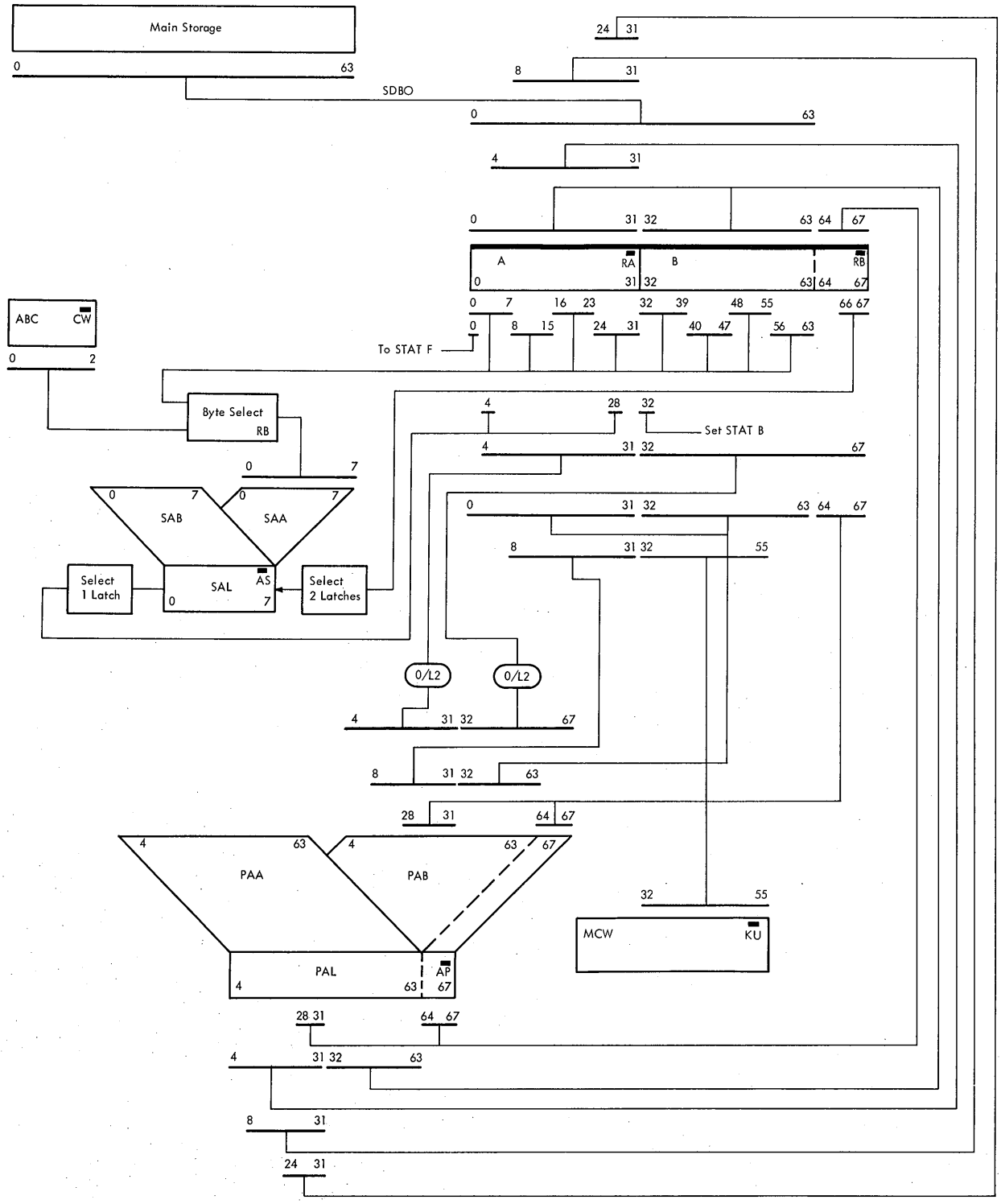


Figure 2-20. AB-Register Data Flow

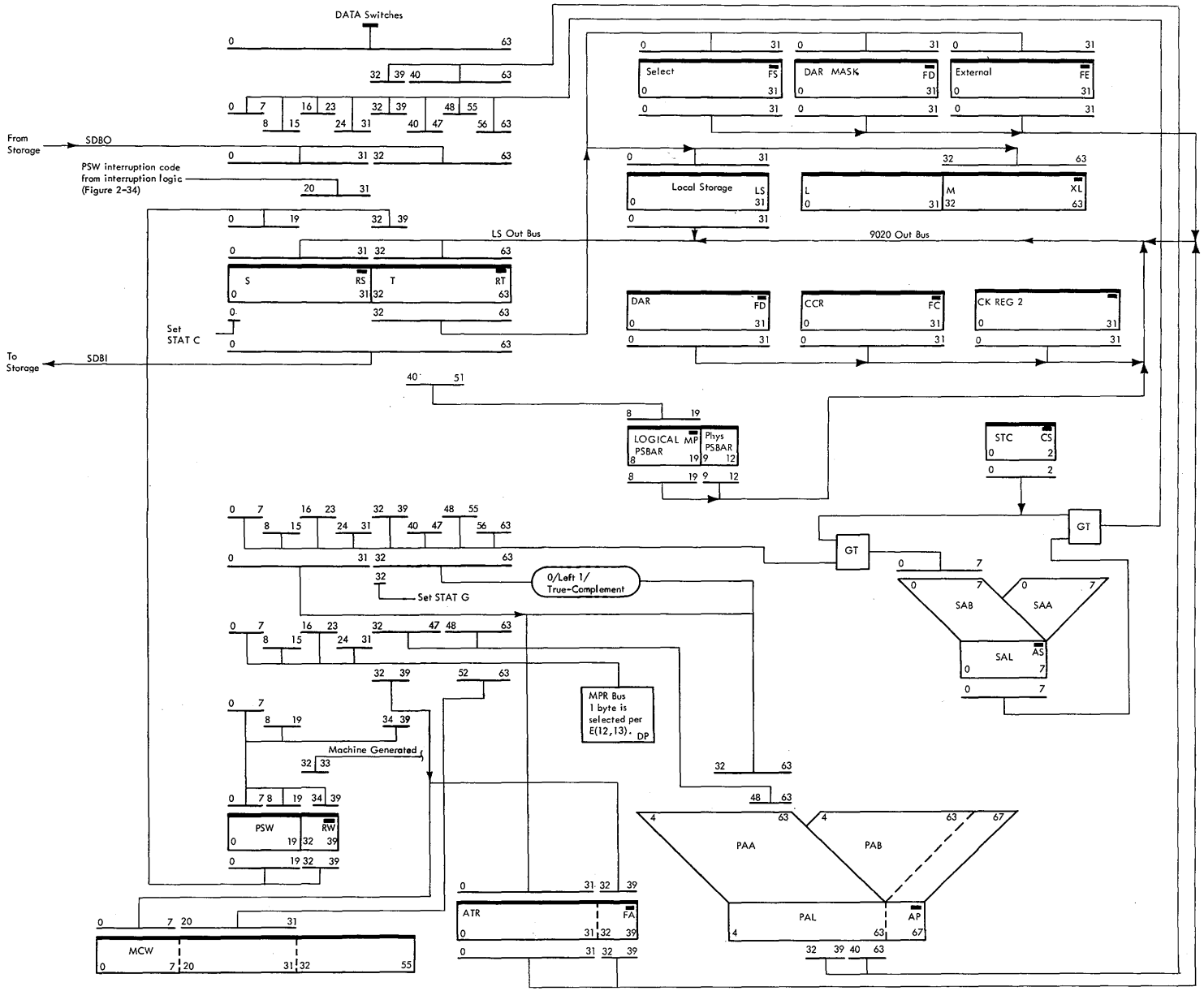


Figure 2-21. ST-Register Data Flow

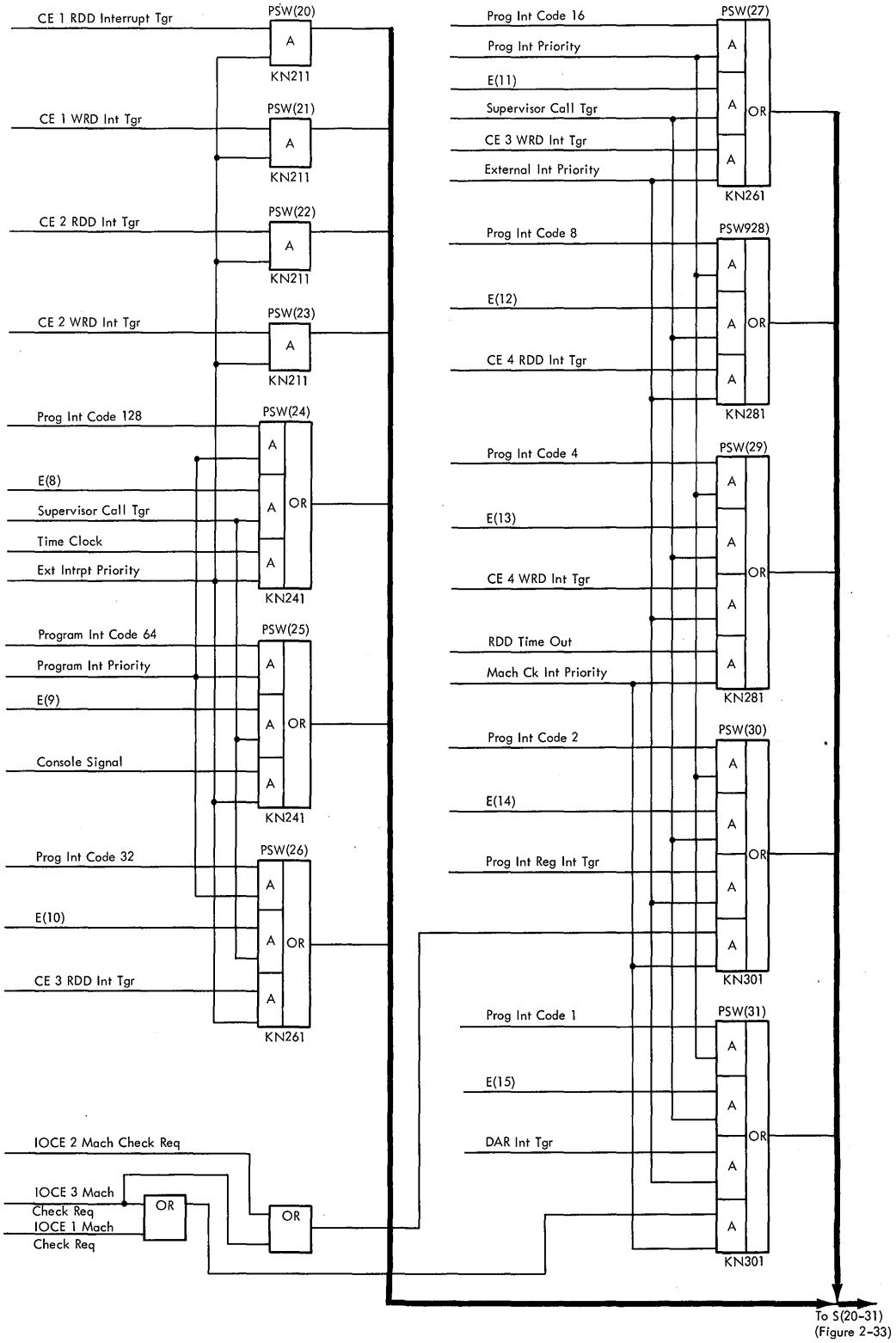


Figure 2-22. PSW Input to S(20–31)

Output

- Outputs are to main storage, LS, PSW register, serial adder, parallel adder, multiply/divide logic, MCW, ATR, Select, DAR Mask, external registers, and M register.

All ST transfer is under ROS control. Transfers to the adders are performed by gate-control triggers. These triggers are reset at P1 clock time of every machine cycle; the specific triggers selected for use are set at P2 clock time of the cycle in which they are to function. (One level of logic delay is incurred in transition, and, as a result, each trigger activates its outgating circuitry at P3 clock time of that same cycle.)

Selection of gate-control triggers for use on any given cycle is controlled either directly by ROS or indirectly by the ST byte counter during ROS-controlled VFL operations. ST transfer to main storage takes place when a storage request is initiated with the 'store' trigger set; 64 data (plus eight parity) bits are then transferred to the SDBI. (Only T-register information can be gated to the LS.)

PSW information from S(0–15) and T(34–39) is transferred to the PSW register under control of ROS microorders. In addition, the following 9020 registers can be loaded from ST: ATR, Select, DAR Mask, PSBAR, and External.

The contents of ST, plus parity, can be transferred to PAA. In addition, T-to-PAA data transfer logic is capable of both true or complement and left 1 shift, and either T(32–47) or T(48–63) can be gated to PAA(48–63).

Byte transfer from ST (for product and quotient insertion during multiply and divide operations) is controlled by E(13–15). (The selected product/quotient bytes are transferred to the MPR bus.)

AB AND ST BYTE COUNTERS

For operations involving the serial adder, it must be possible to extract bytes from doublewords contained in AB and ST and to assemble bytes into ST for subsequent storage. These capabilities are provided by two byte counters: the ABC for controlling AB byte transfer, and the STC for controlling ST byte ingating and outgating. ABC and STC are displayed on roller 6, position 2, bits 24–26(ABC) and 33–35(STC).

AB Byte Counter

- Inputs are from PAL(61–63), T(57–59), E(13–15), and IC(21–23).

- ABC logic increments, decrements, or retains absolute values.

The ABC (Diagram 4-207, FEMDM) consists of three triggers and three incrementer latches. These components are designated T0, T1, T2, and L0, L1, L2, in each group, and represent decimal values of 4, 2, and 1, respectively. Thus, the ABC is capable of selecting any AB byte from 0 to 7. Both the trigger and latch groups are capable of receiving information (000–111 binary); modification (incrementing/decrementing) is performed through the use of incrementer-decoding logic on the input of the incrementer latches.

ROS controls the transfer of information into the ABC from PAL(61–63), E(13–15), and IC(21–23); data from T(57–59) is controlled by scan logic.

In operation, binary values of 000–111 (specifying AB bytes 0–7) are transferred into the ABC triggers at clock time under ROS control [ROSDR(25–30)]. The incrementer-decoding logic samples the ABC triggers and, under ROS control, sets that value, incremented by 1, decremented by 1, or absolute, into the incrementer latches at not-clock time. The incrementer latches are then sampled, and the outputs are decoded into eight lines (0–7) to select one of the eight AB bytes for transfer on the following machine cycle. In addition to controlling register transfer, ABC trigger outputs are utilized by scan operations and certain ROS-branch-decode functions.

Note: E(13–15), or IC(21–23) can also be entered directly into the incrementer latches, at not-clock time, under ROS control.

The ABC triggers are reset with a negative P1 clock pulse at clock time of each machine cycle and set with incoming data at P2 clock time. The incrementer latches are reset with a negative P2 not-clock pulse at not-clock time of each machine cycle, with data transfer occurring at P3 not-clock time.

The contents of the ABC triggers are transferred to the incrementer latches, and the same value is returned to the ABC triggers (regenerated), during each machine cycle in which no other ABC transfer controls are specified by ROS (provided that an 'I-Fetch reset' signal does not occur).

ROS control signals ('000 to ABC' and 'I-Fetch reset') reset the ABC and ABC incrementer to 0 by allowing both the triggers and the latches to reset on the following machine cycle.

ST Byte Counter

- Inputs are from PAL(61–63), T(54–56), E(13–15), and D(21–23).

- STC logic increments, decrements, or retains absolute value.

The STC (Diagram 4-208, FEMDM) consists of three triggers, three bipolar (polarity-hold) latches, and three incrementer latches. The triggers are designated as T0, T1, and T2 and the latches as L0, L1, and L2, representing decimal values of 4, 2, and 1, respectively. Thus, the STC is capable of selecting any ST byte from 0 to 7. The STC triggers (with associated polarity-hold latches) and the STC incrementer latches are capable only of receiving information (000–111 binary); modification (incrementing/decrementing) is accomplished through the use of incrementer-decoding logic on the input of the incrementer latches. The polarity-hold latches retain each STC setting for one additional cycle, providing for a resultant data byte to be gated into ST at the same time the next sequential ST byte is being gated for processing.

ROS controls the transfer of information into the STC from E(13–15), D(21–23), and PAL(61–63). Entry of T(54–56) to the STC is controlled by scan logic.

In operation, binary values of 000–111 (specifying ST bytes 0–7) are transferred into the STC triggers at clock time, with the associated polarity-hold latches assuming the same value at not-clock time of that same cycle. The incrementer-decoding logic samples the contents of the STC triggers and sets that value (incremented, decremented, or absolute) into the incrementer latches. The incrementer latches are then sampled, and the outputs are decoded into eight lines (0–7) to select the ST bytes for transfer during a subsequent machine cycle.

Note: E(13–15), D(21–23), or a defined constant can also be entered into the incrementer latches under ROS control.

The polarity-hold latches are set (or reset) at not-clock time to the value of the STC triggers. This value (specifying an ST byte) is retained in the polarity-hold latches until not-clock time of the following cycle. Thus, at clock time of the following cycle, with the STC triggers having been set to a new value and the incrementer latches possibly containing a modification of this new value, the previous STC-trigger setting is still present in the polarity-hold latches. This retained value now allows information from the serial adder to be placed into the ST byte that was previously transferred out, at the same time that the incrementer latch output is transferring the next sequential ST byte to the serial adder for processing.

All incrementer latch decode lines are sent to the mark-trigger logic (specifying byte areas for main storage entry), and decode lines 0, 3, and 7 are sent to the branch logic controlling the ROSAR setting. An 'STC greater than 3' signal is also transferred to the branch logic controlling ROSAR whenever incrementer latch 0 (binary value of 4) is set.

Gating of the contents of the STC triggers into the incrementer latches and regeneration of that latch value to the triggers are performed each machine cycle in which no other STC ingating controls are specified by ROS (provided that an 'I-Fetch reset' signal does not occur).

The '000 to STC' and 'I-Fetch reset' signals reset all STC triggers and latches on the following machine cycle. The '1 to STC bit 0' and 'I-Fetch reset' (for RR instructions) signals cause the incrementer to assume a decimal value of 4; the '011 to STC' signal sets incrementer latches 1 and 2, thus setting the incrementer to a decimal value of 3.

MARK TRIGGERS

Eight mark triggers (contained in the CE) indicate which bytes of the doubleword on SDBI are to be entered into main storage on a store operation (roller 6, position 2, bits 0-7).

Mark trigger logic (Diagram 4-209, FEMDM) is ROS-controlled; operation is as follows. ROS control field L (ROS sense latch positions 43–46) is decoded to activate one or more of four mark trigger signal lines. These lines set the mark triggers as required: (1) individually, per the STC, (2) in groups of four (0–3 or 4–7), and (3) unconditionally, by setting both the 0–3 and 4–7 groups. ROS micro-orders to set mark triggers also set the 'store' latch to generate a 'store' signal, which is sent to the selected storage unit.

F-REGISTER

The F-register is a one-byte (plus parity) trigger register that is used in certain arithmetic, logical and data-transfer operations (Figure 2-23). (F-register is displayed on roller 1, position 2, bits 0–7).

Input

Inputs to F are under ROS control. All positions involved in an operation are reset at P1 clock time of the same cycle in which they are to receive information, with the ingating occurring at P2 clock time. Data and external control information is received during direct-control read operations, and serial adder outputs are received during VFL operations. F(0–3), F(0–4), and F(4–7) are utilized by Set Key and Insert Key instructions, logical instructions, and decimal multiply and divide instructions. F(6–7), F(0–7), and F(4–6) are used by the Load ID instruction and by the Diagnose instruction (define storage kernal, and store processor interruption kernal operations). CE identity plugs, input to F(6–7), are plugged at installation time, to identify the CEs in the system (on card 01B-A4D3). CE identity bits 6 and 7 (plus a parity bit) are used by the Load ID instruction to indicate to the control program the identity of those CEs attached to this system.

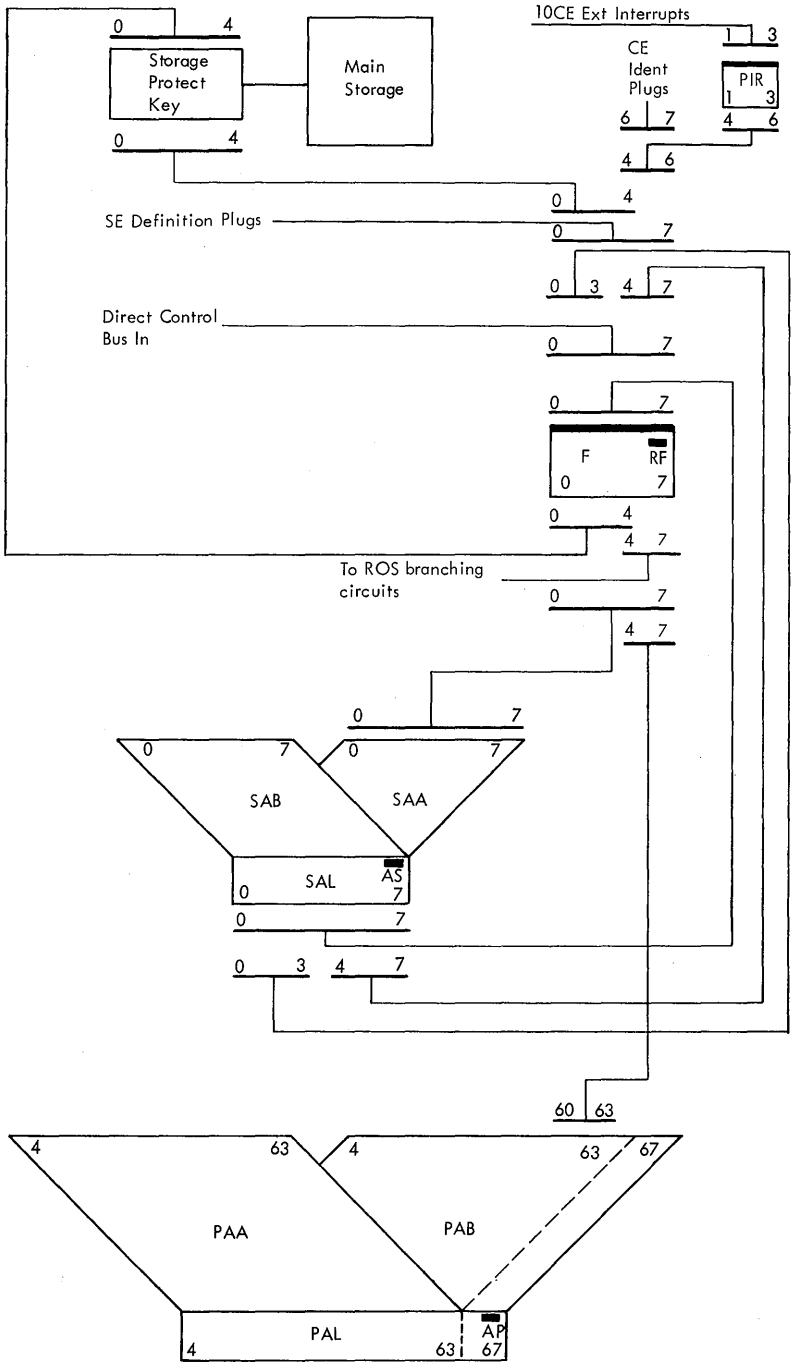


Figure 2-23. F-Register Data Flow

SE definition plugs, input to F(0–7), are plugged at installation time to identify the SEs (and DEs) in the system (card 01B-A4D2). Storage identity bits 0–7 plus a parity bit are used by the Diagnose instruction (Define Storage kernal operation) to indicate the quantity and location of SEs (and DEs) attached to the system.

Input to F(4–6) is received from the Processor Interruption Register (PIR). External interruption requests from IOCE processors are preserved in PIR to be used by the Diagnose instruction (store PIR kernal operation). The external interruption is recognized only when the CE's PSW bit 7 is set to 1 and is processed after execution of the current instruction is completed (with the exception of Delay, Convert and Sort Symbols, Convert Weather Lines, and Repack Symbols instructions, which are terminated.) The interruption causes the old External PSW to be stored at PSA location 18 (hex) and a new External PSW to be fetched from PSA location 58 (hex). The type of the interruption is identified by interruption-code bit 30. The source of the interruption is identified by the PIR contents.

Output

All outputs are under ROS control. F(0–7) is transferred to the serial adder by means of a gate-control trigger at clock time, and F(4–7) is transferred to the parallel adder under control of the parallel adder input logic. F(0–3) is transferred to the storage protect area during set-key operations. F(4–7) is also used in ROS branching.

F(6–7) is gated to local storage during execution of the Load ID instruction. The Diagnose instruction stores F(0–7) in byte 3 of the word following the MCW, during the define storage kernal operation; F(4–6) is stored in the same location during the store processor interrupt kernal operation.

G-REGISTER

The one-byte G-register (0–7) is indicated on roller 1, position 1, bits 19–26. Its only input is from SAL(0–7) (Figure 2-24). This register is the data buffer between an issuing CE and its Direct Control Bus Out. It is used by WRD instructions, which transfer data (CE to CE only). Also, the G-register can be gated via the 9020 Out Bus (24–31) to the LS Out Bus. This path is used during execution of the resident micro-diagnostic routine.

PSW REGISTER

- Inputs are from ST and interruption-control logic.
- Outputs are sent to ST and CE control circuitry.

Although the PSW is 64 bits in length, the PSW register contains only 28 bits (Figure 2-25). The remaining information (generated by the CE at the time of an interruption) is used to identify the cause of the interruption and to allow the CE to return to the correct program address.

PSW register trigger logic is shown in Figure 2-26. The 'gate S(0–7) to PSW(0–7)' signal resets the triggers and, through the logic delay, provides the gating signal to allow the ST register information to enter the PSW register. Thus, the information remains in the PSW register until it is replaced by a new PSW.

All PSW register input and transfer is initiated by ROS micro-orders. When an interruption occurs, a series of micro-orders in the accessed ROS word transfer the contents of the PSW register into ST for subsequent entry into main storage.

The format of the instruction in E (interrupted instruction) is decoded, thus providing the instruction-length code to be entered into the PSW register before transferring the contents of the PSW register to ST. Micro-orders also transfer the old PSW address (generated by the interruption control logic) for that particular interruption to D(17–20) to develop the old PSW address for that interruption. Either

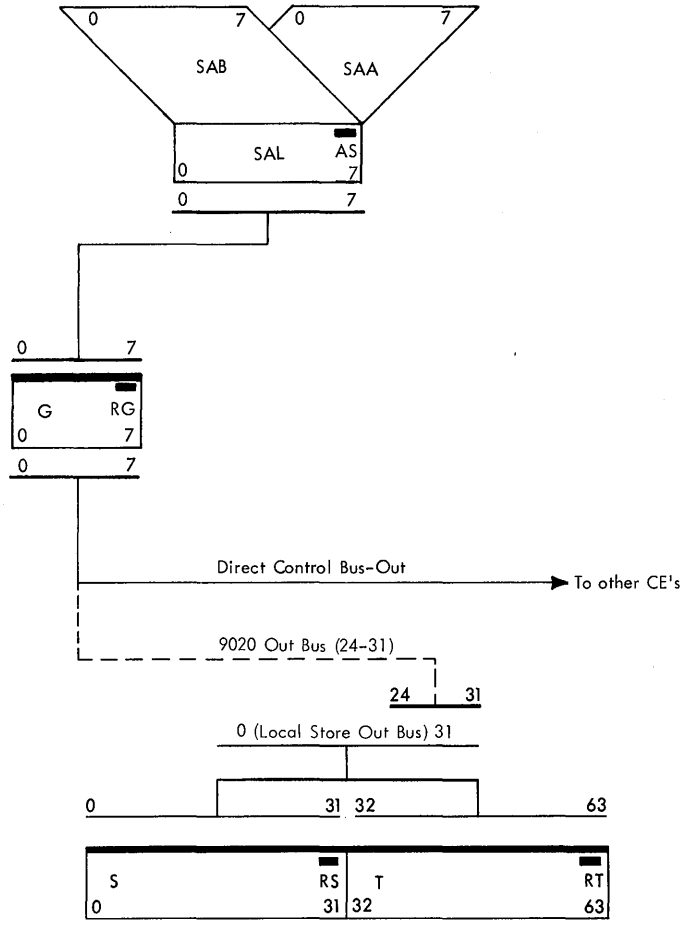


Figure 2-24. G-Register Data Flow

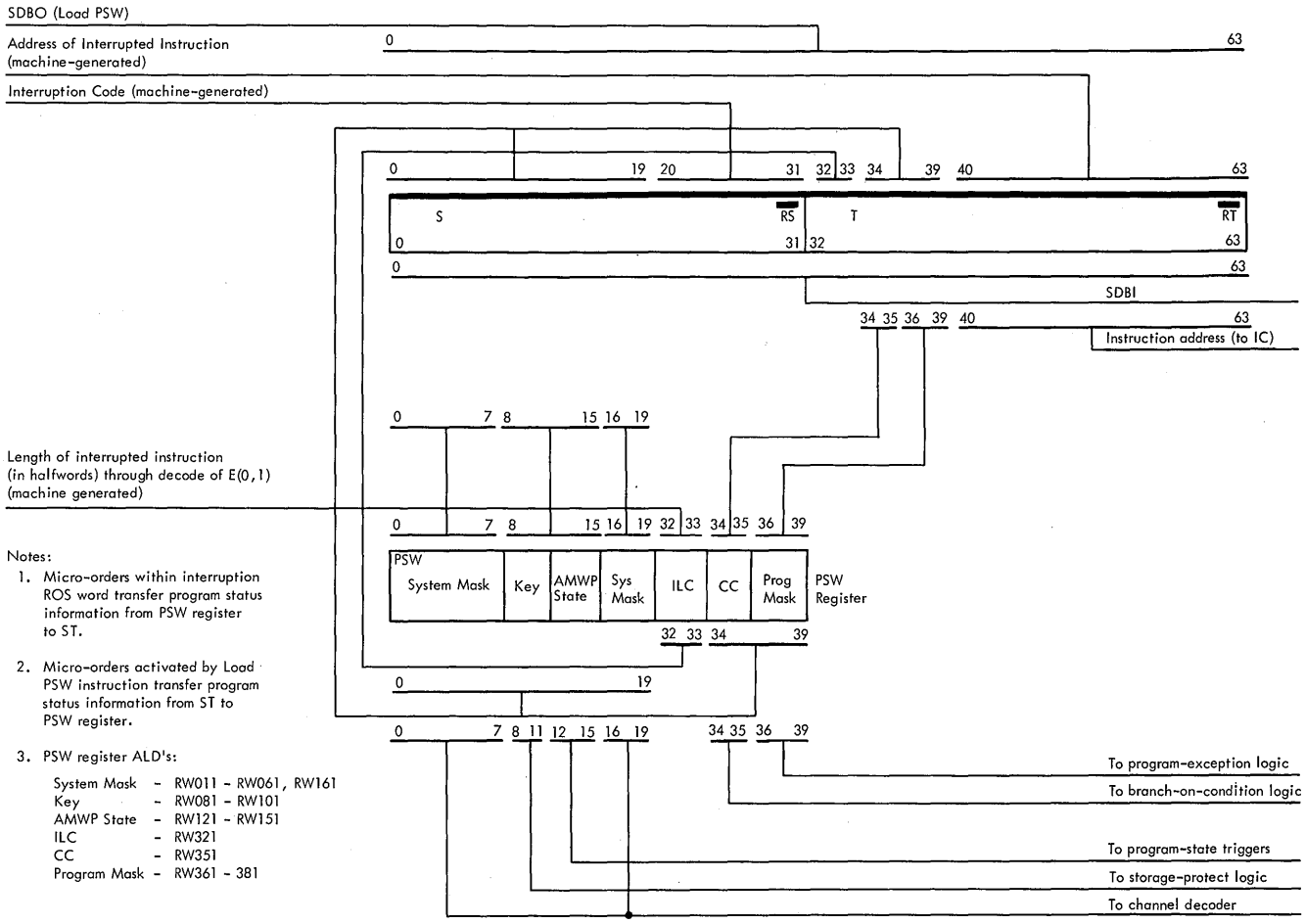
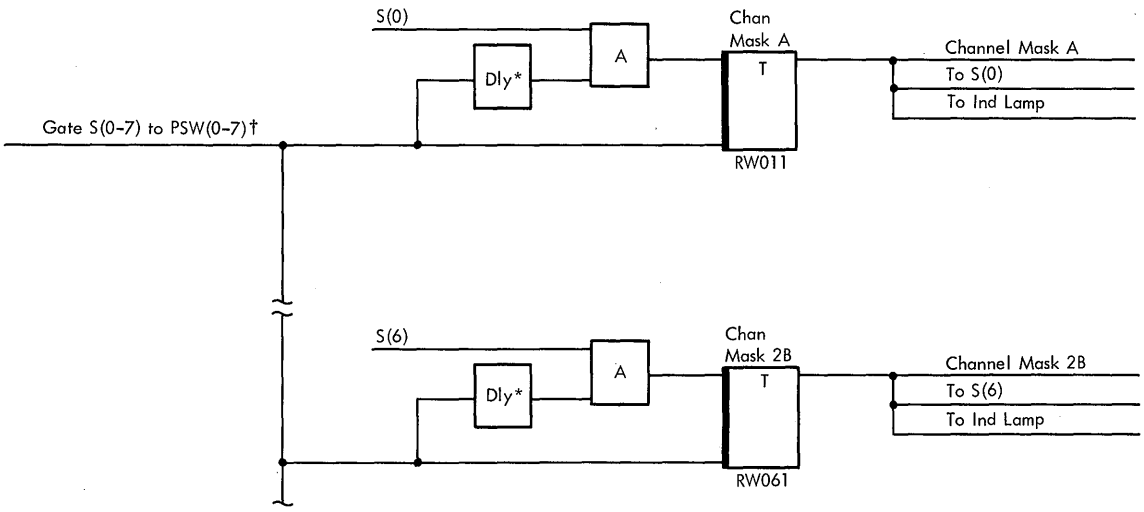


Figure 2-25. PSW Register Data Flow



† ROS Micro-Order

* One level of logic delay to provide the set signal after the trigger is reset.

Figure 2-26. PSW Register (0,6) Logic

8 or 16 (depending on the current instruction address) is subtracted from the IC and inserted into the instruction-address field of the assembled PSW. Remember, however, that IC contains the programmer's address, not the physical storage location, which will be either logical and physical PSBAR combined or an ATR slot and IC(5–11) combined.

Micro-orders executed by the Load PSW, Set System Mask, or Set Program Mask instructions control the transfer of PSW information from the SDBO to ST and the transfer of PSW data from ST to the PSW register and the IC. The old PSW address (contents of D + 64, decimal) is generated in the parallel adder, also under ROS control.

The PSW register does not contain data transfer logic; PSW information is constantly available throughout the CE for use as required.

MCW REGISTER

The MCW register is a 40-bit register that provides program control of scan operations (Figure 2-27). During execution of the Diagnose instruction, FLTs, or ROS tests, MCW (0–7, 20–31) is gated from T (32–39, 52–63), and MCW (32–51) is gated from B (32–51). The bits of the MCW are retained in the MCW register and are decoded by the MCW decode circuitry, to perform the functions as specified in Chapter 4. MCW (21–31) can set values into the address sequencers, FLT scan counter, and FLT clock.

Note: Four groups of MCWs use the same MCW register: (1) FLT, (2) ROS test, (3) Diagnose for the CE when in state 3, 2, or 1, and (4) Diagnose for the CE when in state 0. See Chapter 4, Section 2, for the format of each MCW.

PSBAR

The preferential storage base address register (PSBAR) consists of two registers and an associated counter: logical PSBAR, physical PSBAR, and the PSBAR counter. The function of the two registers is to allow the programmer to place the preferential storage area (PSA) in any 1024- (dec) word block of storage and still access it with the same 1024-word group of addresses, 0000-0FFF (hex). PSBAR is used only for PSA accesses: when a main storage access request is made, and bits 9–19 of the requested address equal 0, then PSBAR (9–19), rather than the 0's, is gated to the storage address bus (9–19). Bits 9–12 are gated from physical PSBAR and 13–19 from logical PSBAR to SAB (Figure 2-29,B). The PSBAR registers may be loaded

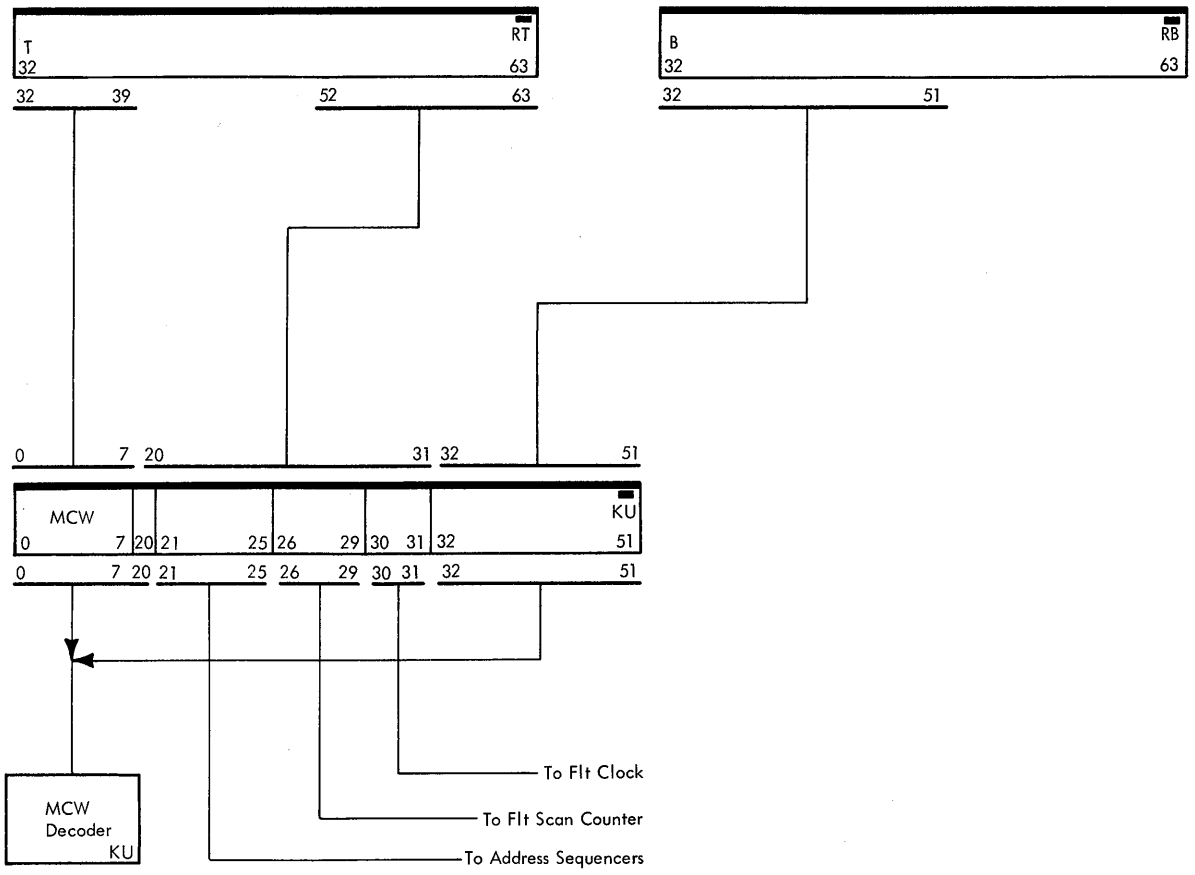


Figure 2-27. MCW Register Data Flow

either by the manual Register Set function (Figure 4-1) or by the Load PSBAR (LPSB) instruction (Figure 2-28,A).

The PSBAR counter enables CE hardware to search for a valid SE by altering the bits in logical PSBAR.

Logical PSBAR. Logical PSBAR (9–19) is loaded via T(41–51) from either the data switches in a Register Set of PSBAR operation or from the location specified by an LPSB instruction. In a PSBAR counter stepping operation (search for valid SE), bits 9–12 are loaded directly from the PSBAR counter (Figure 2-28,B).

Logical PSBAR outputs, bits 9–12 and bits 13–19, are used separately. Bits 9–12 are used in three operations to select an ATR slot from which data is gated into physical PSBAR: Register Set of PSBAR, execution of the LPSB instruction, or when the PSBAR counter is being stepped. Logical PSBAR bits 13–19 are gated to SAB to select a 1024- (dec) word block within an SE for a PSA access (Figure 2-29,B). All of logical PSBAR (9–19) is gated to ST via the 9020 out bus and the LS out bus in the execution of the Store PSBAR (SPSB) instruction (Diagram 5-804).

Physical PSBAR. The inputs to physical PSBAR (9–12) are from the ten ATR slots. The ATR slot to be gated in is selected by bits 9–12 of logical PSBAR for PSBAR counter-stepping. Register Set of PSBAR, or LPSB instruction operation. The output of physical PSBAR (9–12) is gated to SAB, selecting an SE for a PSA access (Diagram 4-602).

PSBAR Counter. Input to the PSBAR counter is from the ATR decode of logical PSBAR bits 9–12. The counter, therefore, has a value of one greater than logical PSBAR bits 9–12 (Figure 2-28,B). The output of the PSBAR counter is gated to logical PSBAR (9–12) when the PSBAR counter is being stepped. This occurs whenever the CE hardware searches for a valid or selected SE in the following operations: system IPL, system PSW restart, external start, or 'step to alternate PSA' (which results from a PSA access error) (Diagram 4-608). In each case, the PSBAR counter (9–12) is gated to logical PSBAR bits 9–12. Logical PSBAR bits 9–12 are decoded to select an ATR slot, which is gated to physical PSBAR (9–12). Physical PSBAR is used

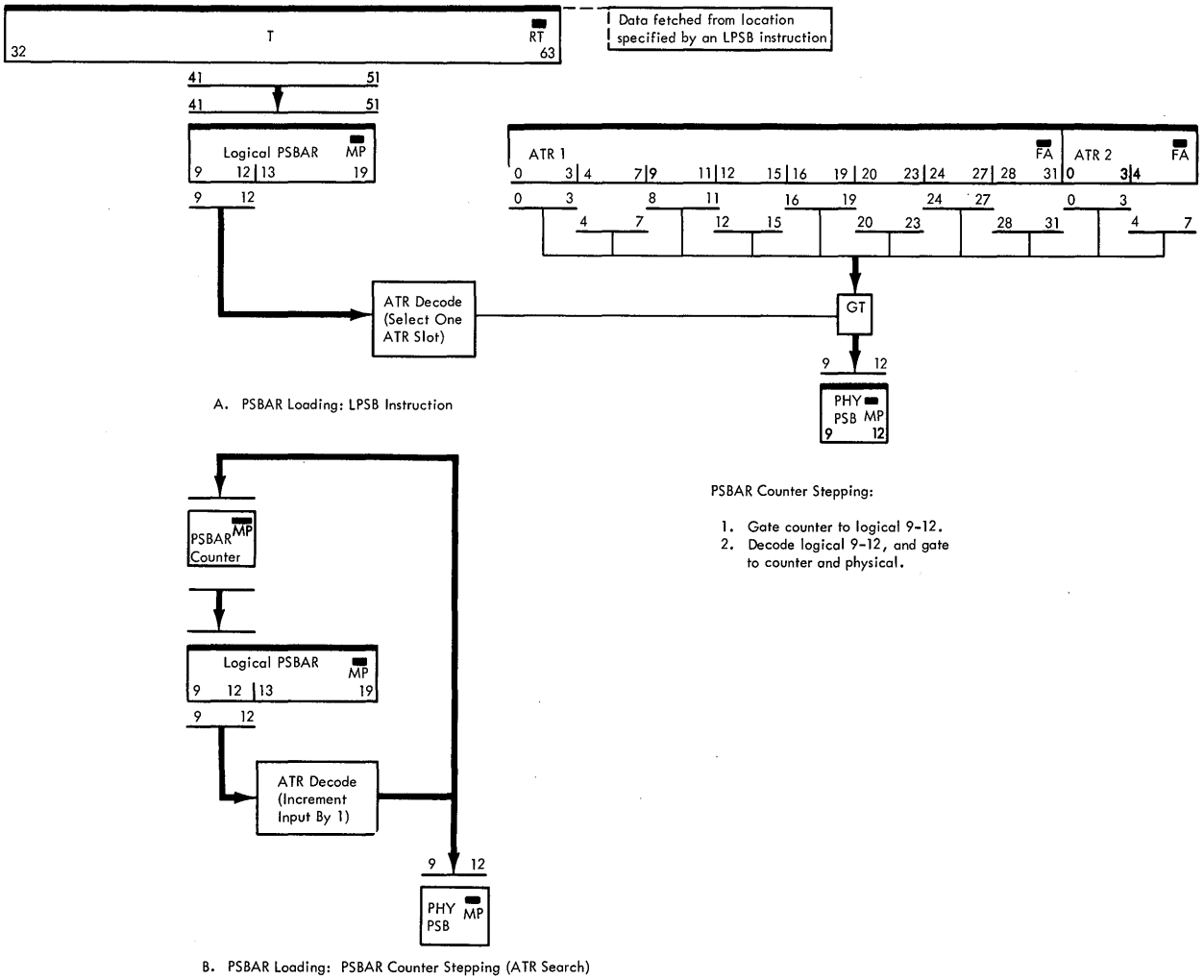
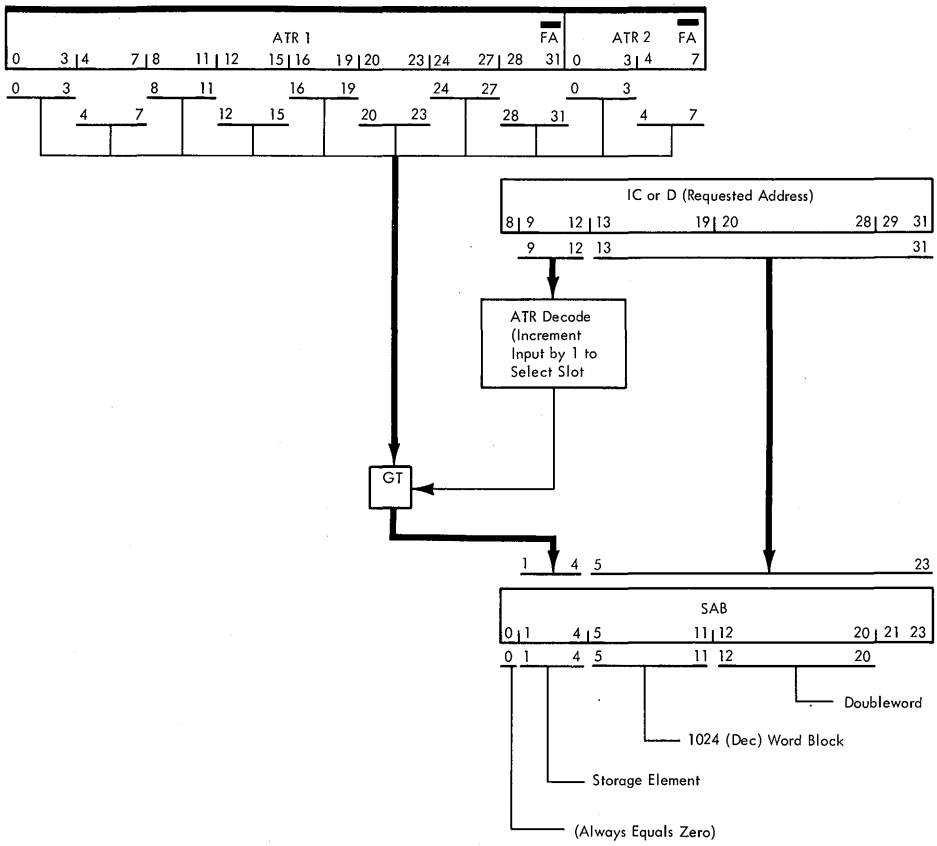
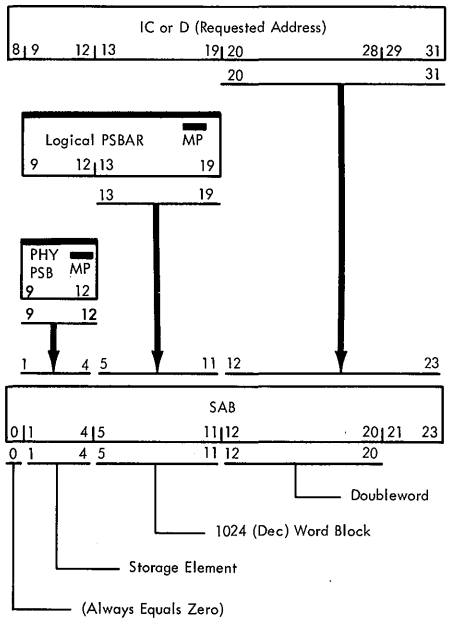


Figure 2-28. PSBAR Loading



A. PSBAR and ATR Data Flow: Normal (Non-PSA) Addressing.



B. PSBAR and ATR Data Flow: PSA Addressing.

Figure 2-29. PSBAR and ATR Data Flow for Main Storage Addressing

in a comparison to determine whether or not it specifies the selected, and/or valid, SE. If it does not, the PSBAR stepping procedure is repeated (Diagram 4-608).

ADDRESS TRANSLATION REGISTER (ATR)

The ATR comprises two registers: a 32-bit (plus 4 parity bits) register ATR-1, and an 8-bit (plus 1 parity bit) register ATR-2. ATRs 1 and 2 can be logically considered as one 40-bit register (Figure 2-29). The ATR provides dynamic address translation of logical address blocks into selectable physical address blocks. Address translation takes place for both normal addressing and preferential storage addressing. ATR is parity-checked.

ATR-1 (0–31) is loaded from S(0–31), and ATR-2 (32–39) is loaded from T(32–39). During address translation, the ATR decoder outgates a selected 4-bit slot from the ATR to the SAB.

DIAGNOSE ACCESSIBLE REGISTER (DAR)

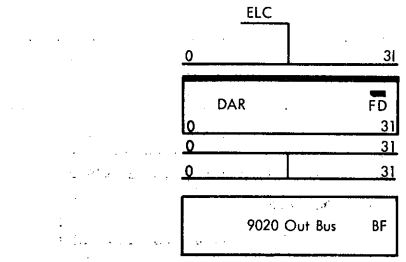
The diagnose accessible register (DAR) is addressable only by the Diagnose instruction. This register is used to store and identify hardware-generated external interruption requests. At the time of an external interruption, the DAR contents are read out to supplement the PSW; bit 31 of the PSW is set to indicate a DAR read-out.

The DAR bit layout is shown in Figure 2-30. When an external element generates one of the specified interruption requests, its identification bit is set in DAR. This bit causes an external interruption, provided the following mask conditions are met:

1. Each bit position of the DAR has a corresponding bit position in the mask register associated with DAR. If the corresponding DAR mask bit is set to 1, and bit 7 in the current PSW is set to 1, an external interruption occurs at the completion of the current instruction.
2. When the CE is in state 2, 1, or 0 (with the TEST switch in the off position), the CE's element check (ELC) signals cannot be masked off; i.e., PSW(7) and DAR Mask bits are ignored. Thus, if the CE is configured to receive a SCON instruction from a CE which is generating an ELC signal, an external interruption is recognized.

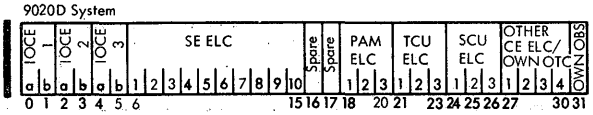
Note that there are only two bits allotted for each IOCE, and they are not sufficient to indicate the necessary conditions. Consequently, these bits are encoded (Figure 2-30) and cannot be considered distinct indications.

The DAR contents are preserved until read out by the Diagnose instruction, at which time the entire register is reset. The Diagnose instruction reads the entire DAR contents and ignores any mask bits that may be set to control interruptions. During the read operation, no new interruptions can be set into DAR.

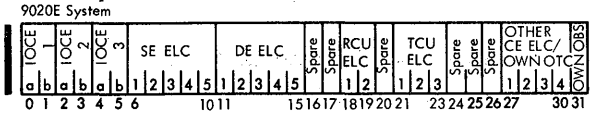


DAR Bit Assignments:

9020D System



9020E System



a	b	
0	0	No Checks Signals
0	1	OBS (Note: Pulse = CCR Parity; Level = OBS)
1	0	OTC
1	0	ELC

Figure 2-30. DAR Data Flow and Bit Assignments

DIAGNOSE ACCESSIBLE REGISTER MASK (DAR MASK)

The DAR MASK (Figure 2-31) is addressable only by the Diagnose instruction, loaded from T(32–63) during execution of the Set DAR Mask kernal. By writing a mask bit pattern into DAR Mask, the Diagnose instruction establishes which elements in the system will be allowed to cause an external interruption in the CE. The DAR Mask output is to T(32–63) via the 9020 out bus and LS out bus, under ROS control.

DAR Mask bit positions correspond to bit positions of DAR. If the DAR Mask position is set to 1 (and PSW bit 7 is set to 1), a corresponding interruption request through DAR sets PSW bit 31 and requests an external interruption. If the DAR Mask bit is 0, the corresponding position of DAR is not allowed to propagate the DAR bit into the PSW.

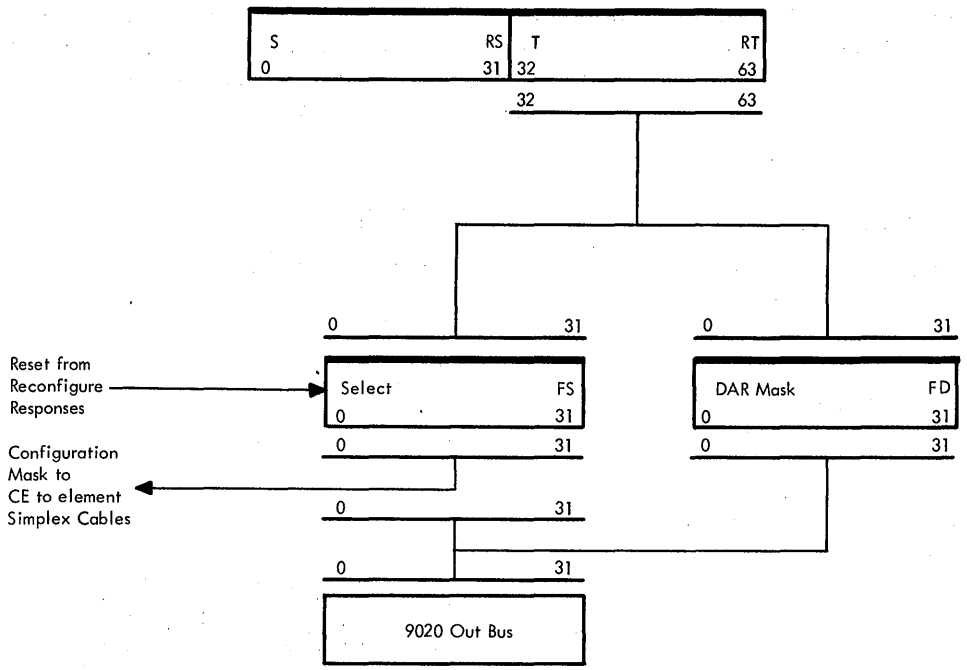
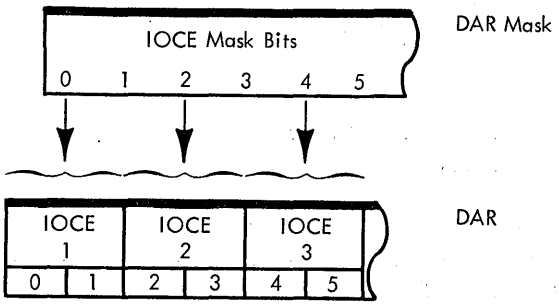


Figure 2-31. DAR Mask and Select Register Data Flow

DAR Mask will mask both bits for each IOCE with a single mask bit. Specifically, DAR Mask bits 0, 2, and 4 are used to mask IOCEs 1, 2, and 3, respectively. (DAR Mask bits 1, 3, and 5 are not used.)



CONFIGURATION CONTROL REGISTER (CCR)

The CCR is a one-word register (32 control bits plus 4 parity bits) used to define the intercommunication between elements in a system (Figure 2-32). It is loaded with the configuration mask bits during execution of a SCON instruction. The SCON instruction may be executed within the CE or by another CE. The CCR in the CE consists of a state field (bits 0, 1), SCON field (bits 2–5), inhibit logout-stop field ILOS (bit 6), and a communication field (bits 8–17, 20–23, and 29–31).

State Field. CCR bits 0 and 1 provide four possible states, which provide the following controls in the CE:

1. State 0 CCR (0, 1) = 00 is used to gate many of the manual control switches on the CE console. The TEST switch is active only when the CE is in state 0; when on in state 0, it sets test on latched, which guarantees that the CE cannot have its state changed by external intervention as long as the TEST switch remains on.
2. State 1 CCR (0, 1) = 01 gates a limited number of the same manual controls as state 0 and is recallable to state 3 if required.

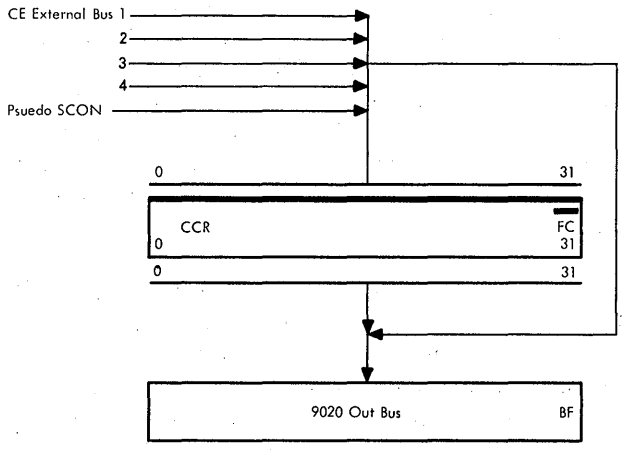


Figure 2-32. CCR Data Flow

3. State 2 CCR (0, 1) = 10 is capable of being recalled to state 3 but is not capable of initiating system reconfiguration.
4. State 3 CCR (0, 1) = 11 is highest operational state capable of initiating system reconfiguration under program control.

SCON Field (CCR bits 2–5) designates which CEs will be allowed to issue a reconfiguration to this CE. One bit is allocated to each of four CEs.

ILOS Field (CCR bit 6) controls issuing of a 'logout stop' signal by the CE to the SE. If the bit is on, the CE will not issue a 'logout' stop signal to the SE when the SE signals a storage check (error gate).

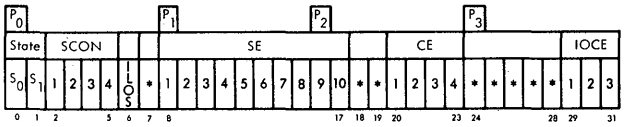
Communication Fields. When a communication bit is set, it enables the interface from the associated unit, which might involve control lines alone or in conjunction with data lines. If the communication bit is not set, the interface from the associated unit is degated, thus providing isolation from elements that are not configured to this subsystem.

Three communication fields are used in the CE: (1) to define SEs (bits 8–17); (2) to define CEs (bits 20–23); and (3) to define IOCEs (bits 29–31).

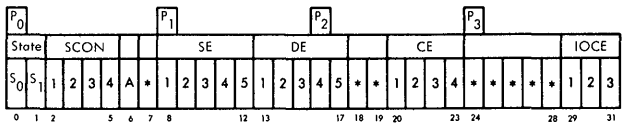
Diagram 4-210, FEMDM shows how the presence or absence of bits in the CCR communication fields allows or prevents interaction between the CE and other elements in the 9020 system.

The CCR formats for the 9020D and 9020E systems are shown below:

9020D Configuration Control Register Format



9020E Configuration Control Register Format



* Denotes unused bits which may be 1 or 0.

Input

Input to the CCR is via the CE external bus in; it consists of the configuration mask bits gated during execution of the SCON instruction. An additional input, called pseudo-SCON is a hardware-controlled function initiated, in System mode, by the LOAD (IPL) or PSW RESTART pushbutton, with the interlock key on.

Output

The contents of the CCR may be gated to ST (32–63) via the 9020 out bus. The CE control logic and communication lines fed by the CCR are shown in Diagram 4-210, FEMDM.

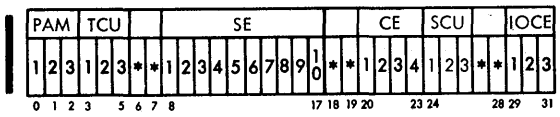
SELECT REGISTER

The select register is a one-word register (32 data bits plus 4 parity bits) that defines the elements to be configured by the SCON or the SATR instructions (Figure 2-31).

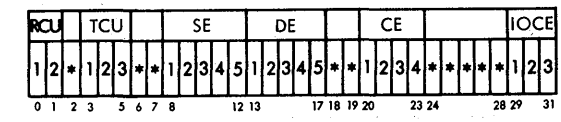
Input

During execution of the SCON instruction, the select register is loaded with the selection mask from the T-register. The select register formats for the 9020D and 9020E systems are shown below:

9020D Selection Mask Format



9020E Selection Mask Format



*Denotes unused bits.

Output

The Select Register bits, which have been set to 1 by loading a selection mask, are gated to CE-to-Element simplex cables or multiplex lines by ANDing 'timing gate' trigger with Stat G for SCON Select: Thus, external elements are selected by the contents of the select register during execution of the SCON instruction.

The responses from the elements that had their mask bits set in the select register are received via the Element-to-CE simplex cables or multiplex lines to reset the corresponding bit in the CE's select register. The contents of the select register are then gated, via the 9020 out bus, to ST for checking. If one or more bits is still on (any 1's), a condition code 2 is set to indicate that one or more elements did not respond (Diagram 4-213, FEMDM).

PROCESSOR INTERRUPT REGISTER (PIR)

The processor interrupt register (Figure 2-23) is a three-bit register used to preserve external interruption requests from IOCE processors.

Input is from the IOCE-to-CE simplex receivers.

Output is gated to bits 4–6 of the F-register.

The Diagnose instruction (Store PIR CE Diagnose Kernal operation) can cause the PIR contents to be stored in byte 3 (bits 28–30) of the word following the MCW; the PIR contents are then reset. (See Diagnose Diagram 5-609.)

EXTERNAL REGISTER

The external register is a one-word (32 data bits plus 4 parity bits) register that specifies the unit and channel address, as well as I/O operation codes, i.e., normal I/O operations, I/O operation for FLT, IPL, and I/O processor operations. This register is also used for transfer of data to other system components during execution of the SCON and SATR instructions.

The bit position assignments for the external register are shown in Figure 2-33. Data flow for the external register is shown in Figure 2-34.

Input

During IPL and FLT operations, inputs to the external register are from the LOAD UNIT switches and from T. During normal operations and during SCON and SATR operations, inputs are from T(32–63). During I/O and I/O processor operations inputs are from T(32–63) and from E(5–7).

Output

The external register output goes to the 9020 out bus and to the CE external bus out.

CHECK REGISTERS

Two check registers are provided (check register 1 and check register 2) for presentation of check indications from CE checking circuits.

Fault detection circuits set a unique bit in the check registers. Figure 2-35 shows the bit assignment for each position of the check registers. Both check registers are displayed on the CE control panel, and their contents are included in the logout information.

Whenever an appropriate bit is set in check register 2, the CE issues an ELC signal to all other CEs. Then, if PSW bit 13 is on, the CE logs out and performs a machine check interruption. If PSW bit 13 is not on (masked off), the logout and machine check interruptions are deferred until software masks the bit on again. Manual controls may modify the internal CE operations upon the occurrence of a malfunction when the machine is in the Test mode. When the CHECK CONTROL switch is in PROC position, the ELC signal is always issued when a malfunction is detected, unless a hard stop condition occurs and the INHIBIT CE HARDSTOP switch is active.

Execution of the Reset Checks CE Diagnose Kernal (by the Diagnose instruction) will cause check registers 1 and 2 in the issuing CE to be reset. (See Diagnose Diagram 5-609.)

DISPLAY REGISTERS: LM, MIXER, XY, K, and N

These registers are used during execution of the display instructions by the 9020E System. (One exception is the K-register which is also used by the DE-Wrap Kernal operation of the Diagnose instruction.)

The subsequent paragraphs describe the basic operations and data paths used by the display registers. Specific register applications during execution of the display instructions are described in Chapter 3, Section 10.

LM Register

The LM register is a doubleword (64 data bits plus 8 parity bits) used during execution of the Convert Weather Lines (CVWL) instruction and the Repack Symbols (RPSB) instruction.

Input

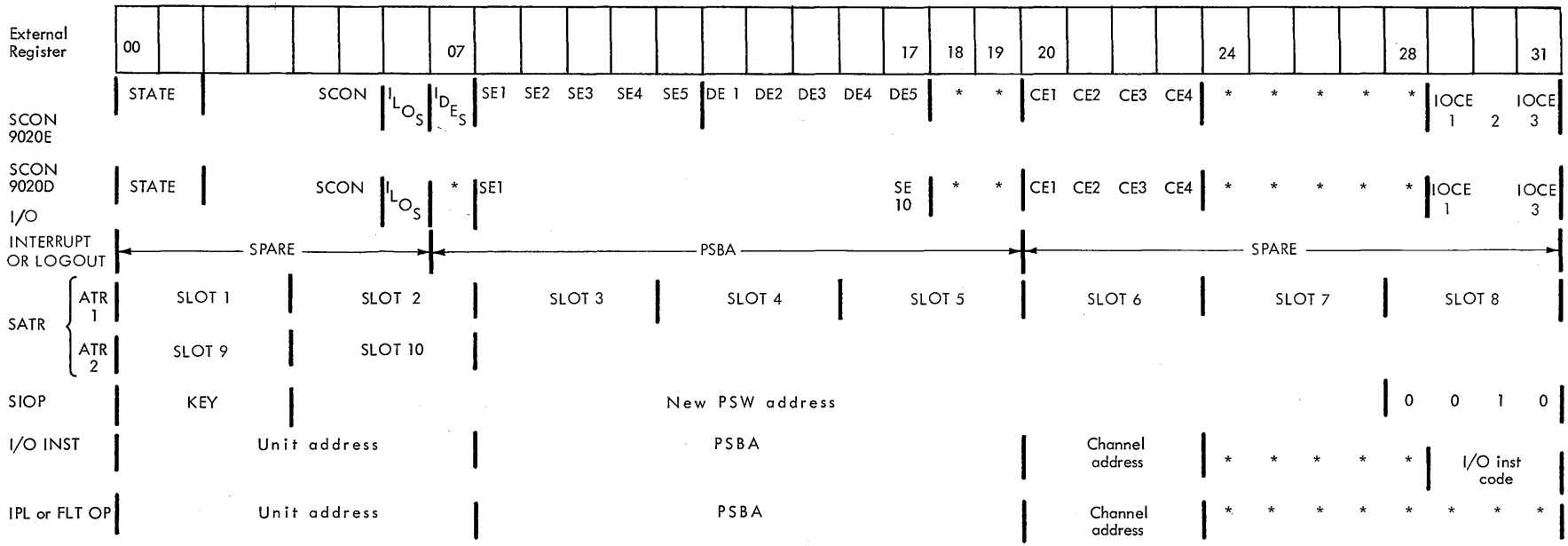
The inputs to the LM register (Figure 2-36) are from storage (SDBO) and from the T-register.

Output

The LM register provides output to the N-register and to the XY register (via the Mixer). The mixer operation is described below.

Mixer

During execution of the Convert Weather Lines and Repack Symbols instructions, it is necessary to transfer data



*not used

I/O INSTRUCTION	Op Code	Ext Reg code 29		31
SIOP	9A	0	1	0
SPCI	9B	0	1	1
SIO	9C	1	0	0
TIO	9D	1	0	1
HIO	9E	1	1	0
TCH	9F	1	1	1

Figure 2-33. External Register Bit Position Assignments

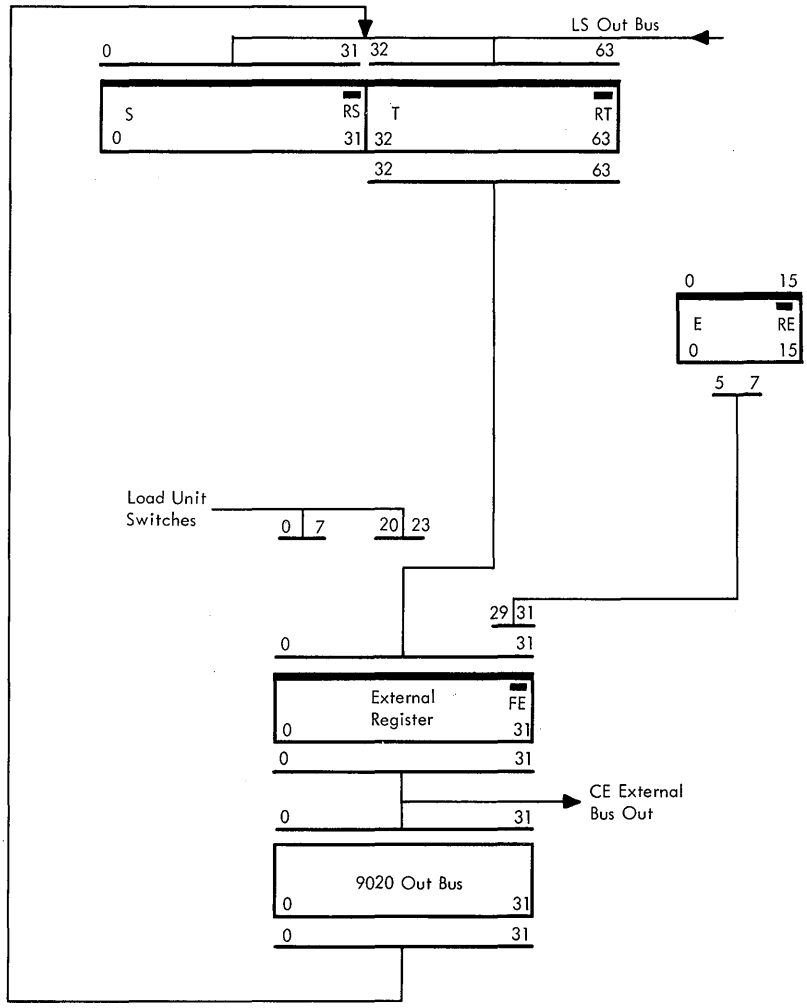


Figure 2-34. External Register Data Flow

between various storage areas and to reformat this data as it is transferred. This is accomplished by moving the data from the LM register to the XY register, via the Mixer.

Diagram 4-211, FEMDM, shows the LM-to-XY reformatting of data via the Mixer. The Mixer ANDs the output of the ROS field (ROSDR 92–96) (decoded as 'format old', 'format new', or 'format weather') with the contents of E(13–15) to generate the proper gates for transferring the desired bit positions from the LM register to the desired bit positions in the XY register. For 'format weather', only E-register bits 14 and 15 are examined since there are only three format weather lines micro-orders. (See Table 2-1.)

Since reformatting involves splitting bytes, parity must be examined as data is moved from the LM to the XY register (so that good parity is generated in XY). The XY parity prediction logic is shown in Figure 2-37 and in Diagram 4-212.

XY Register

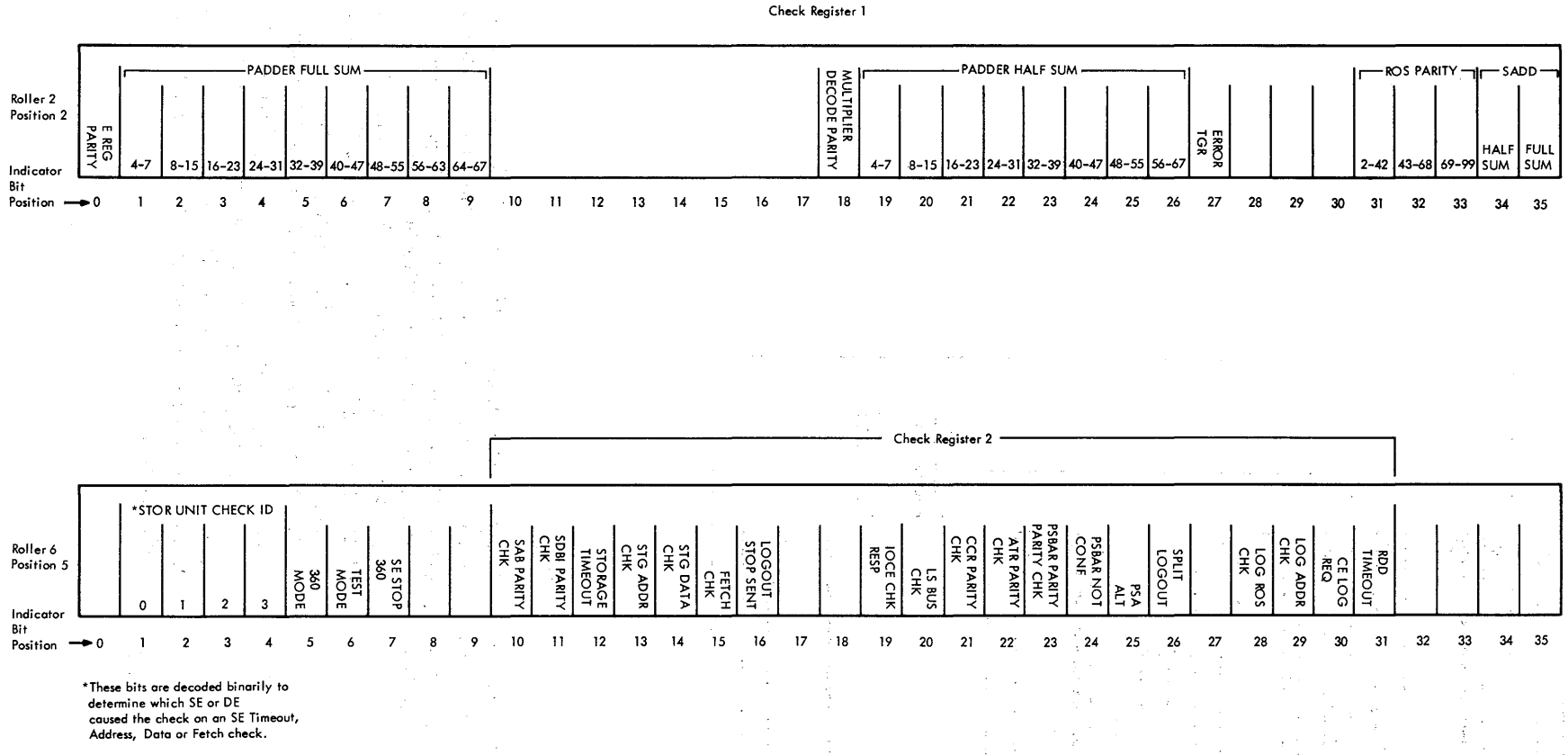
The XY register is a doubleword (64 data bits plus 8 parity bits) register used during execution of Convert Weather Lines and Repack Symbols instructions (Figure 2-36).

Input

Input to the XY register is from the LM register via the mixer.

Output

Output from the XY register is to storage via the SDBI.



● Figure 2-35. Check Register Bit Assignment

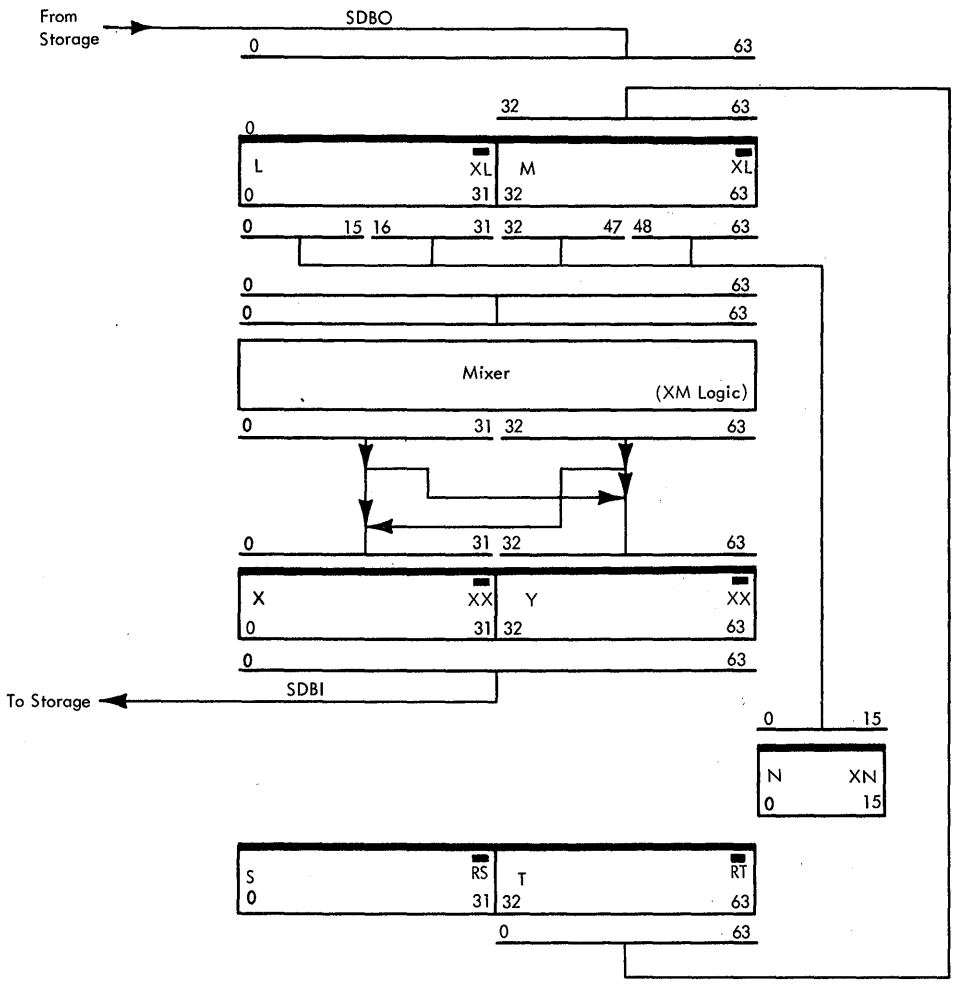


Figure 2-36. LM and XY Registers, Data Flow

K-Register

The K-register is a one-word (32 data bits plus 4 parity bits) register which is used during execution of the display instructions (CSS, CVWL, and RPSB) and during DE wrap test operation. The K-register data flow is shown in Figure 2-38.

Input

Input to the K-register is from PAL(32–63) and from the DE wrap bus.

Output

The K-register provides an output to the parallel adder PAA(32–63).

DE Wrap Bus

A distributed simplex bus (DE wrap bus) is provided on each CE to allow testing of the system display elements (DE). This operation is available via the Diagnose instruction while the CE is in state 0 (Diagram 5-609).

The Wrap operation provides a facility which allows a CE to attach to a DG interface of a selected DE and, once attached, to simulate a DG by accepting data from the DE.

The DE wrap bus is a 16-bit bus from the DE which is gated alternately to K(0–15) and K(16–31). Prior to being gated to the K-register, good parity is generated within the CE (parity is not sent by the DE). For a more detailed discussion of the DE wrap bus, refer to Chapter 4, Section 3 of this manual.

N-Register

The N-register is a halfword (16 data bits plus 2 parity bits) register used during execution of the Repack Symbols instruction. The N-register data flow is shown in Figure 2-39.

Input

Inputs to N are under ROS control. The contents of the LM register, in two-byte groups, can be gated to N per IC (21,22). SAL(0–7) can be gated to N(8–15).

Output

Output from N is gated, under ROS control, to the serial adder B-side from either N(0–7) or N(8–15).

Table 2-1. Format Micro-orders

(Format Micro-orders for Convert Weather Lines)

FMTW- These micro-orders are used when assembling the CVWL output doubleword. A format word is assembled in T, T is gated to M, and, from there, the format word is gated through the mixer into XY by one of the FMTW micro-orders. This is done three times (first using FMTW-0, next FMTW-2, and, finally, FMTW-1) in order to gate one complete output doubleword into XY.

Micro-order	Description
FMTW-0	Gates DA, DL, BL, BR, S and Symbol, CO, C1, ΔYS, and ΔXS bits from M to their assigned positions in output doubleword.
FMTW-2	Gates ΔY2 and ΔX2 coordinates from M to their assigned positions in output doubleword.
FMTW-1	Gates Y1 and Y2 coordinates from M to their assigned positions in output doubleword.

(Format Micro-orders for Repack Symbols)

FMTO- These micro-orders are used when history or current data is moved from Old Refresh memory to New Refresh memory.

Micro-order	Description
FMTO-0	Used to gate current data. Gates LM to XY with no changes.
FMTO-1	Used to gate current data and change target 1 and target 2 to history data. Gates LM to XY and turns off BR1 bit (5).
FMTO-2	Used to gate history data and change both target 1 and target 2 to history data. Gates LM to XY and turns off BR1 and BR2 bits (bits 5 and 7, respectively).
FMTO-4	Gates all target 2 data from LM into positions in XY to make it target 1 and drops target 1 data. Resets P2 bit (P2 bit on indicates to display that target 2 is to be displayed).
FMTO-5,6	Same as FMTO-4, but resets target-2 BR bit in the process (makes it history data).

FMTN- These micro-orders are used when new data (from CSS execution) is moved from a sort bin to New Refresh memory during execution of Repack Symbols instruction.

Micro-order	Description
FMTN-0	Used when sort bin word is in M only. Turns on P2 bit. Gates CSS target from M reg into XY output target-2 positions. Gates nothing into target-1 positions in XY.
FMTN-1	Used when sort bin words are in L and M both. Gates L target to target-1 positions in XY and M target to target-2 positions in XY. Sets P2 bit to 1.
FMTN-2	Used when sort bin word is in L only. Gates L target to target-1 positions in XY. Resets P2 bit in the process. Gates nothing into the target-2 positions in XY.
FMTN-6	Used when sort bin word is in L only. Gates L target to target-2 positions in XY and sets P2 bit. Gates nothing to target-1 positions in XY.
FMTN-7	Used when sort bin word is in M only. Gates M target to target-1 positions in XY and resets P2 bit. Gates nothing into target-2 positions in XY.

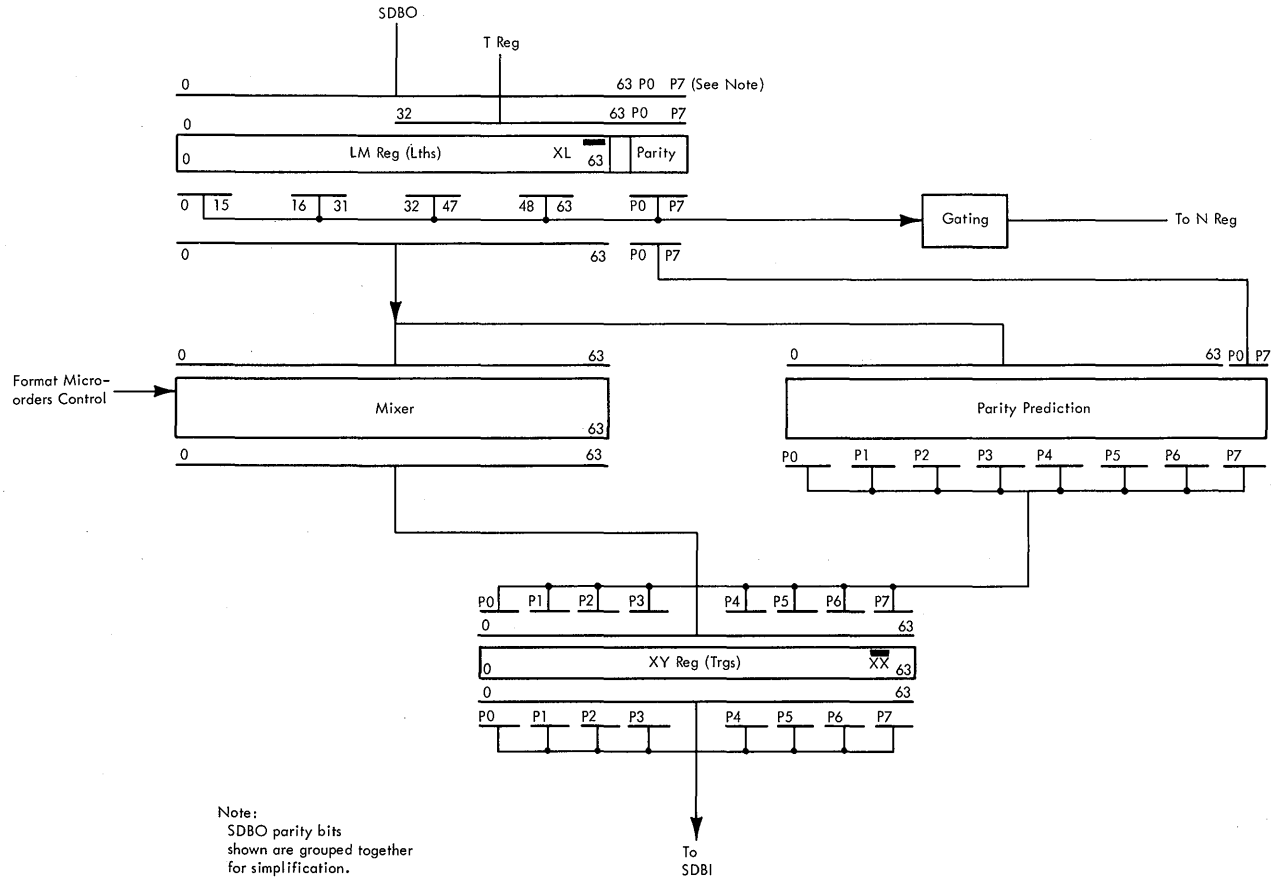


Figure 2-37. XY Register Parity Prediction Logic

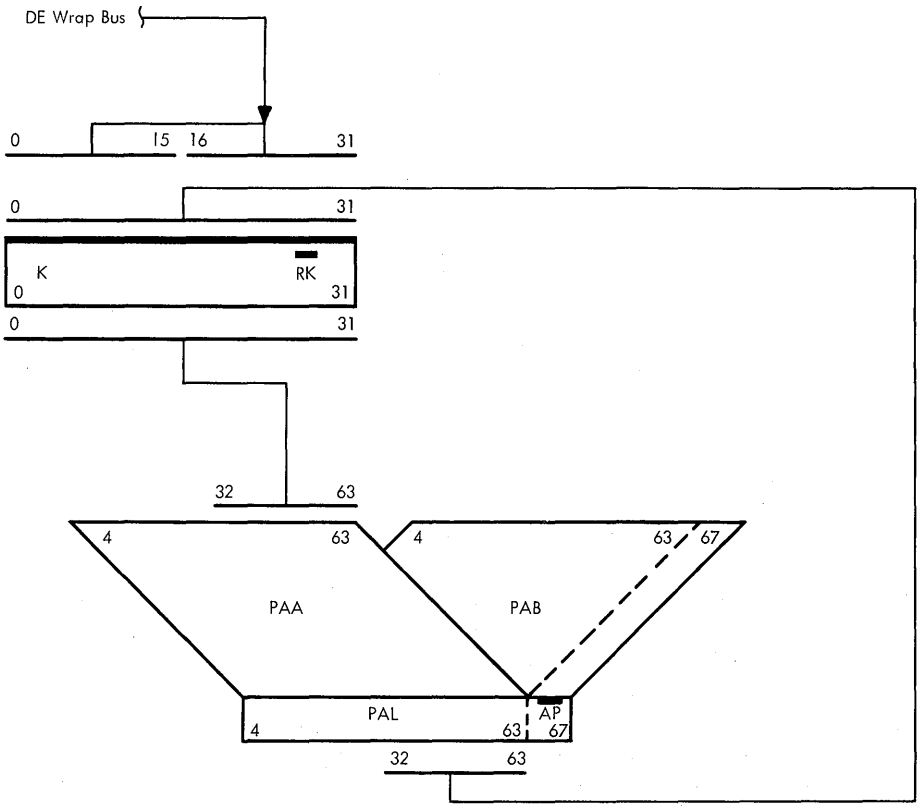


Figure 2-38. K-Register Data Flow

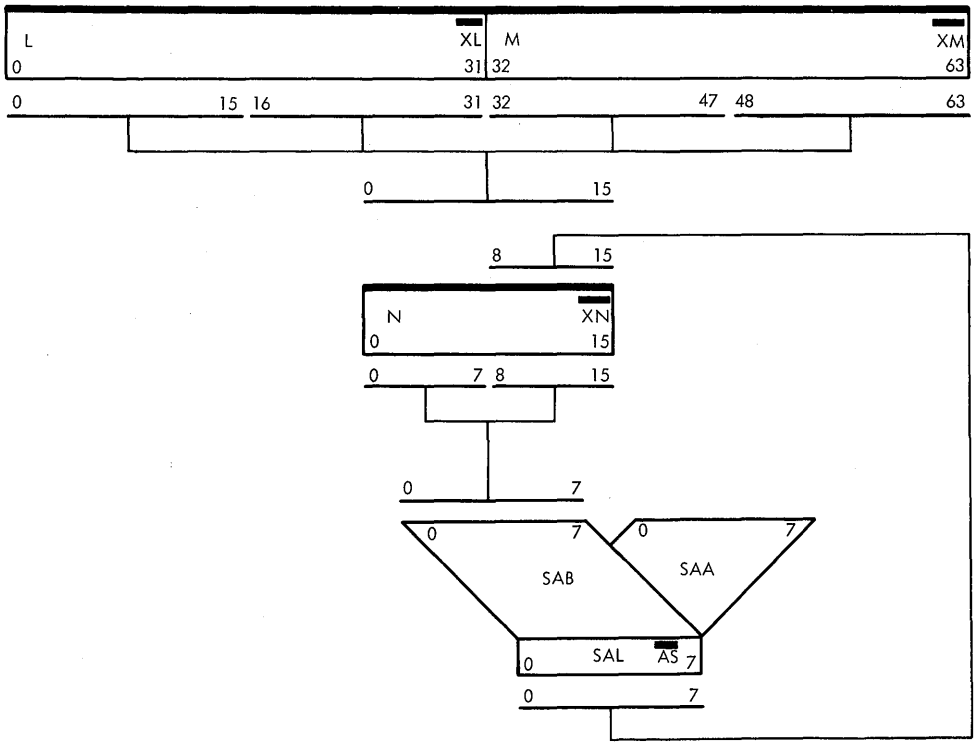


Figure 2-39. N-Register Data Flow

This section describes the operation of the 25-register local storage (LS).

ADDRESSING AND DATA FLOW

- Five-position address registers [LAL (Read) and LAR (Write)] address LS.
- Input to LS is from T only; output is sent to S and/or T, under ROS control.

Two five-position LS address registers [LAL (Read) and LAR (Write)] select the 25 individual LS registers (Figure 2-40). The LS address is received from Q, R, or E under ROS control (or directly from ROS control words when addressing the LSWR). The particular register (Q, R, or E) and field within that register to be set into LAL is determined by decoding ROSDR(36–42). The four-bit LS addresses are gated into the four low-order positions of LAL from Q, R, or E fields. These four-bit addresses are capable of directly addressing registers 0–15 (general-purpose registers). For floating-point operations (requiring the use of registers 16–23), a 1-bit is forced into the high-order position of LAL upon decoding of the floating-point op code. This action increments the four-bit address from Q, R, or E by 16, thus forcing the use of floating-point registers 16–23.

Note: Floating-point instructions are restricted to the use of even LS addresses 0, 2, 4, and 6. Automatic incrementing of these values by 16 then generates LS addresses of 16, 18, 20, and 22.

Long-operand floating-point op codes also force a 1-bit into the low-order position of LAL, in addition to the high-order 1-bit forced by all floating-point op codes. This additional bit further increments the 16, 18, 20, and 22 floating-point addresses by 1, thus generating the second (R1 + 1) register address required for long-operand (64-bit) instructions.

For operations requiring use of the LSWR (register 24), the ROS words controlling these operations force 1-bits into the two high-order positions of LAL [LAL(0,1)]. This action generates a binary address of 24 and is the only means of selecting the LSWR.

Selection of the Q, R, or E field to be entered into LAL is determined by decoding ROSDR(36–42) of the controlling ROS word or by selecting 'NEOP' or 'BEOP' micro-orders, depending on the next programmed instruction. Regardless of the address source or of whether a read LS or write LS operation is indicated, the contents of the addressed register are always read out onto the LS data bus. If a read LS operation is indicated, decoding of ROSDR(10,11) of the controlling ROS word gates the contents of the LS data bus into S, T, or both S and T. When a write LS operation is indicated, the contents of the addressed LS registers are gated out onto the LS data bus in the same manner, but the output resulting from decoding ROSDR(10,11) remains inactive and does not condition the ST ingating controls. A 'write into LS' signal, resulting from the ROSDR(36–42) decoder, then gates the ST bus (T-data) into the addressed LS register.

Information into the LS data bus can come either from the LS, or from the 9020 out bus. When information is gated from the 9020 out bus, the '9020 reg to LS out' line is activated to block normal readout per LAL. (Since, at this time, LAL contains all 0's, a LAL readout would access LS register 0.) The LS data bus gating logic is shown in Figure 2-41 and in Diagram 4-302, FEMDM.

During execution of two operational kernals used by the Diagnose instruction, logout local store registers (FD2) and store DAR (FD4), and when the select register is gated to T during a SCON or SATR instruction, it may be necessary to generate parity bits for one or more of the four bytes being gated to S and/or T since the information in DAR, DAR Mask, check register 2 or the select register may be in either even or odd parity.

The 'local store bus check' latch is blocked from setting, and each byte on the local store bus is checked for an odd count. If parity is odd, a parity bit is not needed, but, if the count is not odd, a parity bit is generated to provide odd parity for each byte to be gated to S and/or T.

The 'generating LS bus parity' line is controlled by a micro-order and is inactive except in the above situations, thus allowing normal parity-checking when gating other 9020 registers via the local store data bus (Diagram 4-303, FEMDM).

DATA TRANSFER CONTROLS

The following paragraphs describe the LS logic involved in read LS and write LS operations. Diagram 4-301, FEMDM, illustrates the read/write logic of LS register 0 and also the common control circuit timings for each 200-ns LS cycle. (Registers 1–24 are identical with register 0.)

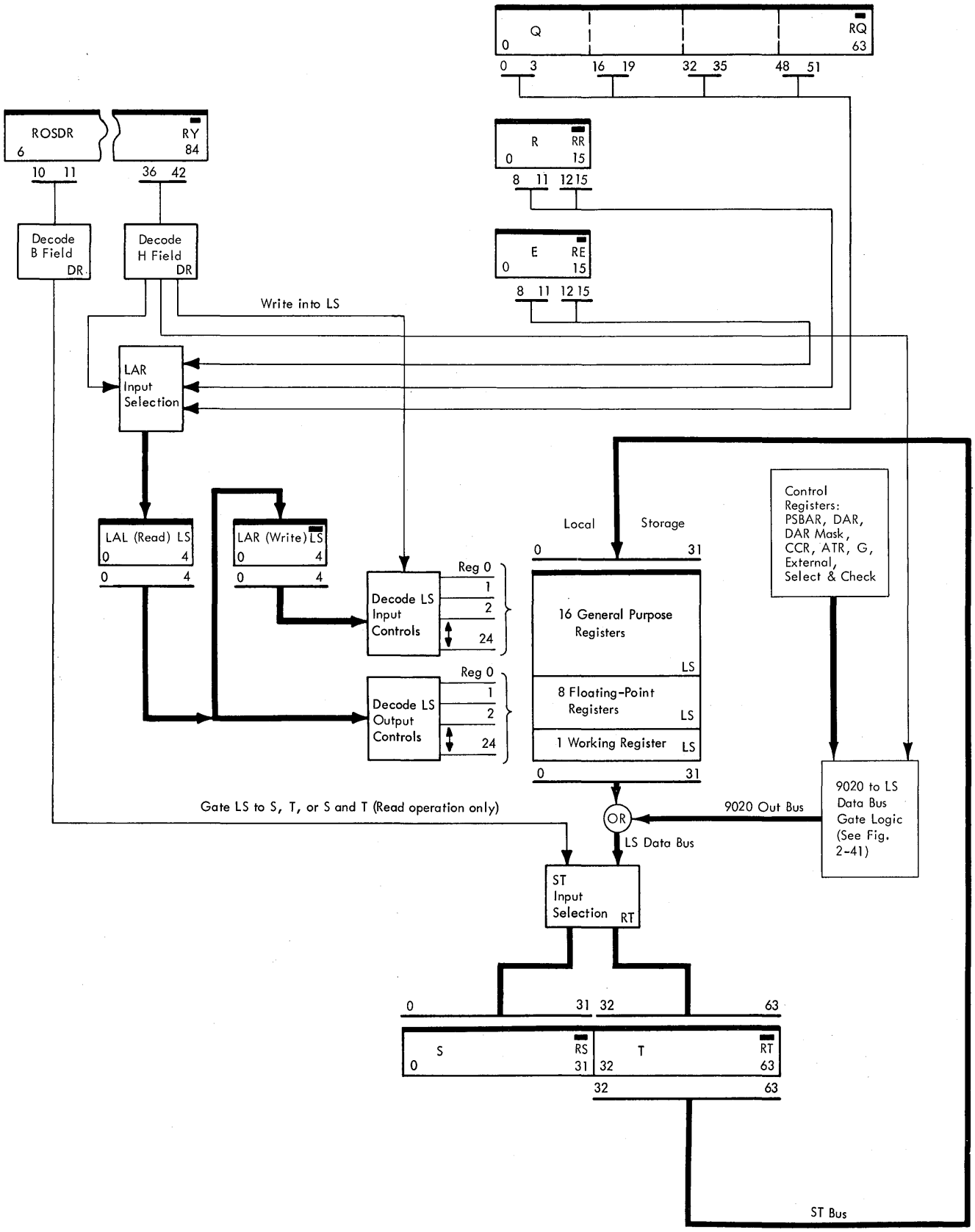


Figure 2-40. Local Storage Data Flow

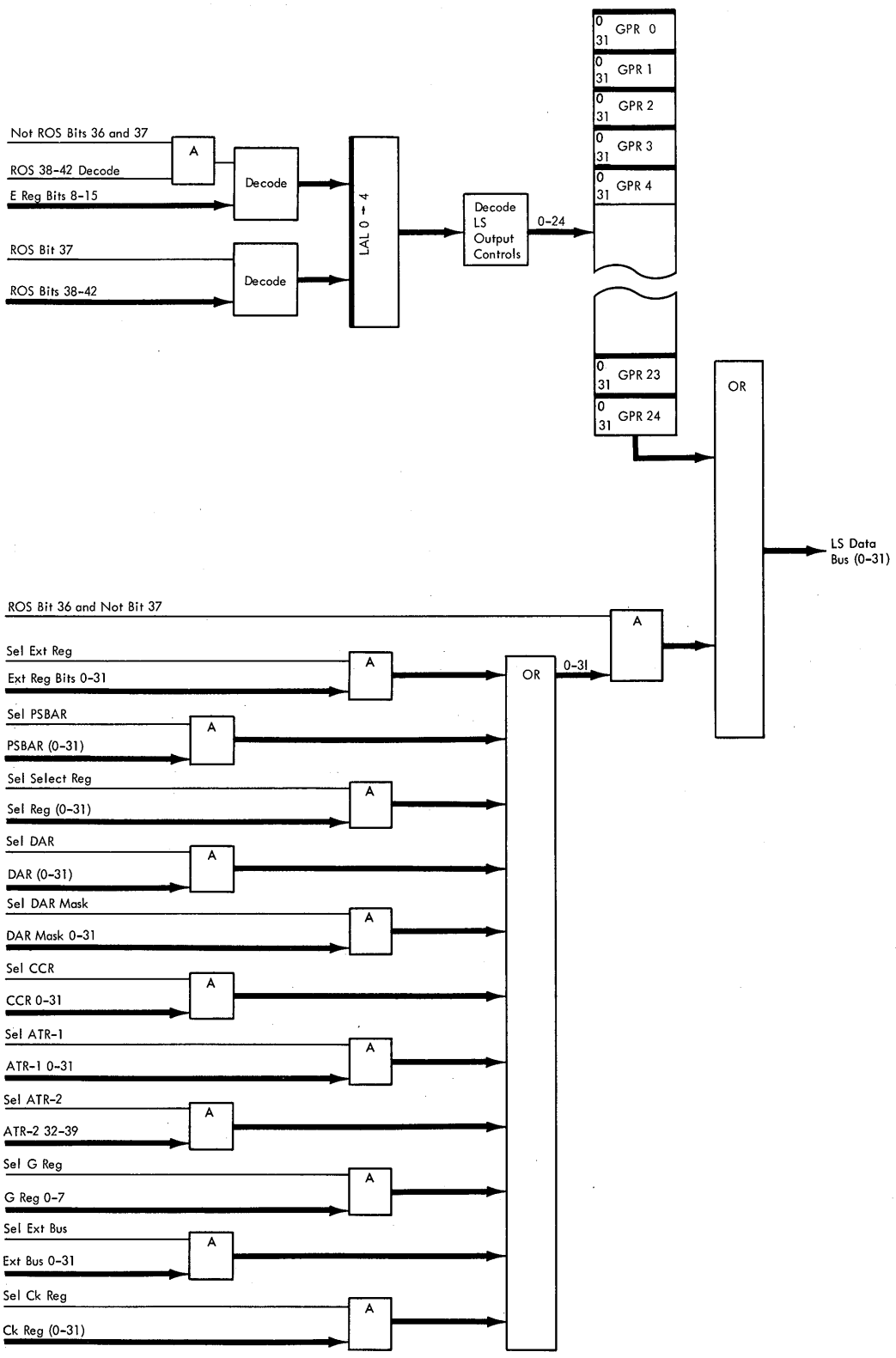


Figure 2-41. 9020-to-LS Data Bus Gate Logic

Local storage addressing and all LS register-operating logic are implemented in 10-ns circuitry; the polarity-hold latches and associated input logic are implemented in 30-ns circuitry.

Read LS Operation

- ROSDR (36–42) sets LAL from specified Q, R, or E field.
- LAL gates contents of selected LS register to LS data bus.
- ROSDR(10,11) gates LS data bus into ST.

Read LS operations are initiated at not-clock time when LAL (Read) is set with Q, R, or E information, as determined by ROSDR(36–42). The LSWR address and the floating-point register address bits are also entered into LAL at this time, depending on ROSDR(36–42). The contents of LAL (Read) are decoded, and the decoder outputs gate the contents of the selected register polarity-hold latches to the LS data bus. ROSDR(10,11) then activates the required ST input logic, and, at clock time of the following cycle, the LS data bus information is set into the ST triggers. Polarity-hold circuits provide nondestructive readout, eliminating the need for regeneration. (Refer to the timing chart in Diagram 4-301 for relative control timings.)

Note: The contents of LAL (Read) are transferred to LAR (Write) every cycle, but no addressing is performed unless a write LS micro-order is decoded in ROS.

Although LS data is available to the CPU approximately 100 ns after the setting of LAR, consecutive LS data readout is limited to 200 ns. LS cycles are therefore defined as being 200 ns long.

Write LS Operation

- ROSDR(36–42) sets LAL (Read) to specified Q, R, or E field.
- LAL (Read) contents are transferred to LAR (Write).
- LAR (Write) decoder selects specified LS register.
- ROSDR(36–42) gates ST bus data into selected LS register.

On write-LS operations, LAL (Read) latches are set at not-clock time with the specified Q, R, or E information. At the beginning of the following cycle, LAL (Read) is transferred into LAR (Write) in the same manner as for a read operation. The selected LS register is also gated to the LS data bus as in read operations; up to this point, read and write operations are identical. (On write LS operations, however, LS data bus information is not gated into the ST register.)

At the beginning of the following cycle, LAL (Read) is transferred into LAR (Write). Further decoding of ROSDR(36–42) generates a 'write into LS' signal that sets the 'write LS' trigger at P0 time. This trigger provides the signal to gate the ST bus (T-data) into the selected LS register at not-clock time of the following cycle. (Refer to the timing chart in Diagram 4-301 for relative write control timings.) Negative levels on the ST bus represent 1-bits and set the respective polarity-hold latches; positive levels represent 0's and reset the respective polarity-hold latches.

For an 'insert sign' micro-order when the result sign is minus, T(32) is forced to a 1. To preserve proper parity for this operation, the parity bit for T(32–39) is inverted before it is transferred to LS.

There are situations when writing into LS must be inhibited. When such a situation occurs, the 'SPEC' (K31) micro-order causes a set signal to the 'inhibit LS write' trigger so that the LS positions remain unchanged.

Note that a minimum of 40-ns coincidence must exist between stable ST bus data (1's or 0's) and an active 'gate T to LS' polarity-hold-latch control signal to give correct data entry. When the 'gate T to LS' signal is deactivated, polarity-hold latches of the selected register remain in their present state until new data is entered on a subsequent write LS operation.

Note, too, that LS data is not parity-checked until it enters an adder at a later time.

LS Timing

Separate address registers for reading and writing coordinate the LS timing to other CE functions. The 'gate LS reg n' signal is generated by LAL (latch timing) to make the LS data available at clock time for entry into a register. LAR, at not-clock time, allows the data entered into T early in the cycle to be entered into LS late in the cycle. Refer to Diagram 4-301, which illustrates the entry of new data into T (reflected by the shift in the ST bus data line) and the subsequent storage of that data into LS.

SECTION 5. SERIAL AND PARALLEL ADDERS

This section describes the operation and application of the serial and parallel adders.

SERIAL ADDER

The serial adder (8 data bits plus 1 parity bit) processes data in binary or decimal format, performs logical AND, OR, and Exclusive-OR functions, and assembles multiply/divide results.

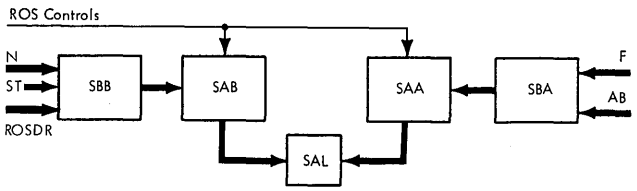
Data flow for the serial adder is illustrated in Figure 2-42. Note that data entered into the A-side of the adder (via final bus-A) comes from either AB or F under ROS control.

Data entering the serial adder is in true or complement form. For a true add operation, the data is entered directly; for a complement add operation, the input data to the serial adder A-side (SAA) is inverted (Figure 2-43).

The 2's complement value is achieved by forcing a hot-1 to the input logic for serial adder latch (SAL) position 7.

Input and Output

Inputs to the serial adder A-side (SAA) are the contents of F or a selected byte from AB (per the ABC); the input to the serial adder B-side (SAB) is a selected byte from ST (per the STC). A bus arrangement transfers data from the registers to the serial adder as follows:



During transfer from the serial adder bus A (SBA) to the SAA or from the serial adder bus B (SBB) to the SAB, ROS controls can alter the data being transferred. The SAA input can be altered by the following functions: decimal excess-6, complement add, shift, crossgating (interchanging of incoming bits 0–3 with bits 4–7), and zone and sign insertion. The SAB input may be altered by the following functions: sign insertion, special digit insertion, and special gating for changing destination (for example, placing bits 0–3 into bit positions 0–3 and 4–7).

After the sum has been developed and placed into SAL, gating signals from ROS allow the information to be transferred to F, to G, to N, or to a selected byte in ST.

Adder Operation

A simplified summary of serial adder operation is shown in Figure 2-44. SAA and SAB are combined in the serial adder to produce a bit-carry, a bit-transmit, or a half-sum. A bit-carry is developed when both input bits are present, a bit-transmit when either input is present, and a half-sum when only one of the input bits is present (Figure 2-45). Carry-in and half-sum conditions combine to produce a full sum. The table in Figure 2-45 shows the conditions that produce a full-sum bit. For example, if SAA is a 1-bit and SAB is a 0-bit, there will be no bit-carry, but a bit-transmit and a half-sum will be produced. If no 'carry in' signal is present, the full-sum is a 1.

The 'carry in' signal is developed by the carry lookahead logic. A test is made for a carry from the next-lower position or for a carry developed from bit-transmits and a lower-order carry (Figure 2-46). The carry lookahead logic saves time by providing an immediate carry rather than using another cycle to ripple a low-order carry through the adder.

Accurate results are achieved by parity-checking and parity-correction circuits. Tests for error conditions are made at half-sum and full-sum levels as well as on decimal input data.

Controls

- Selected data enters on SBA and SBB.
- Data is first modified on transfer from SBA to SAA and from SBB to SAB.
- Final modification occurs as data enters SAL.

There are three control areas: input bus, final bus, and SAL (Figure 2-47).

ROS sense latch 86, field R, selects F or AB as a data source for SBA. If the latch is set, F is selected; if it is reset, AB is selected. Gate control triggers provide the gating signals to transfer data to the buses. For SBA, if F is not selected, an AB gate control trigger is selected by the value in the ABC (Diagram 4-401, FEMDM). Similarly, the STC

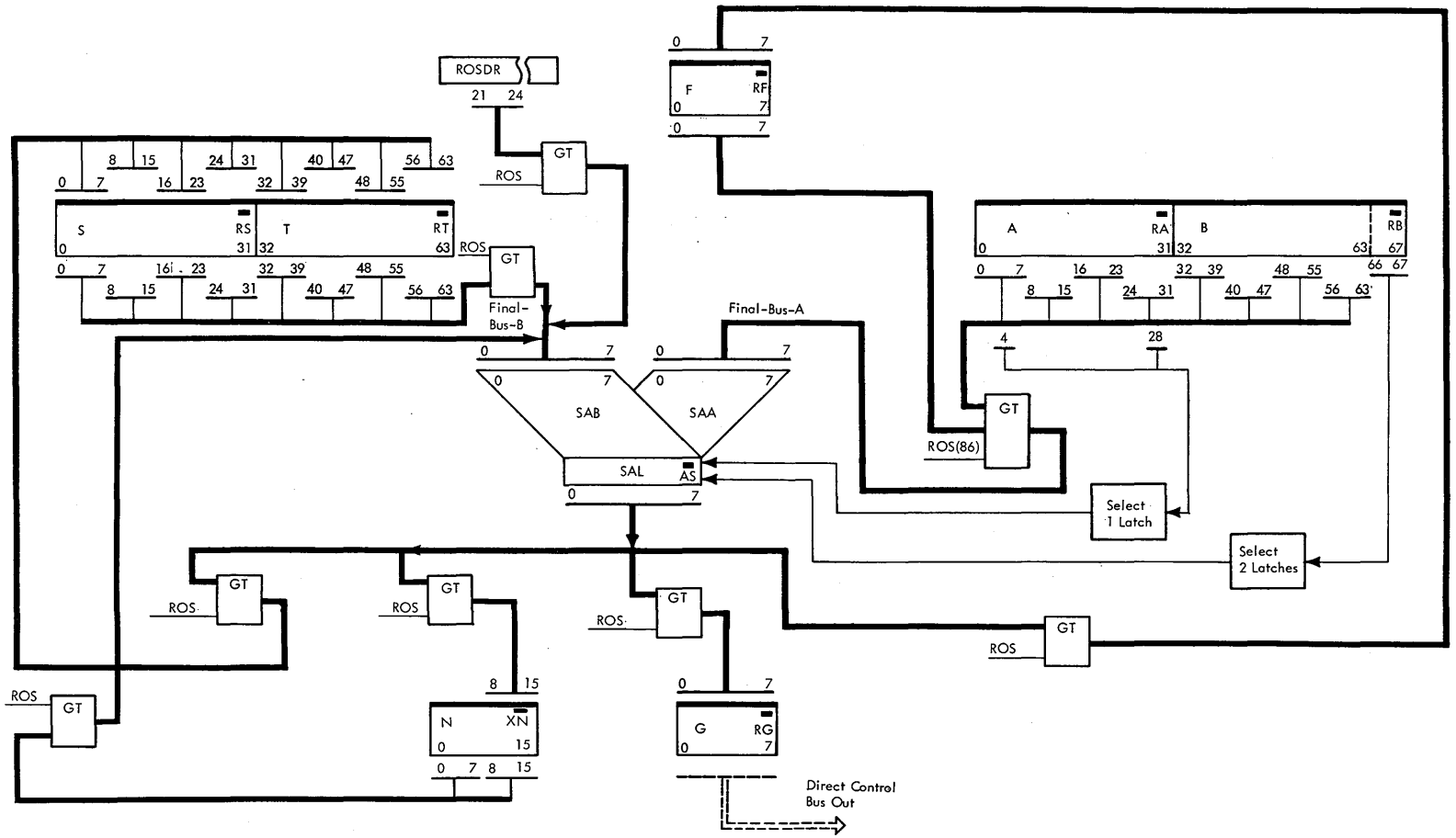


Figure 2-42. Serial Adder Data Flow

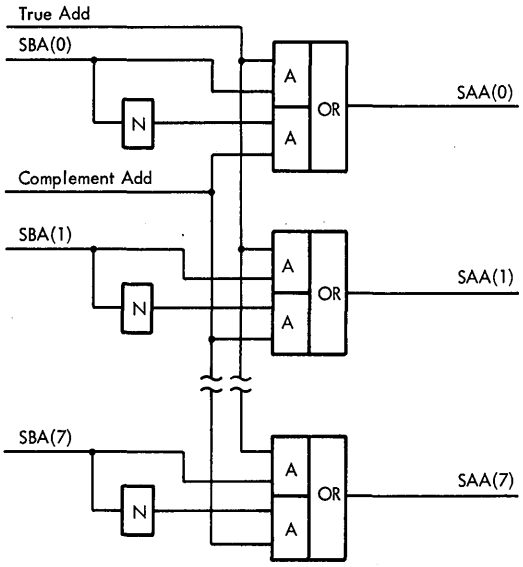


Figure 2-43. True-Complement Data Entry

selects an ST gate control trigger. One byte of data is selected by each gate control trigger, and two gate control triggers are selected (one for SAA and one for SAB) each machine cycle whether or not the data will be used.

ROS fields: M(bits 69–73), N(bits 74–77), E(bits 21–24, 81), C(bits 12–16), and W(bits 2–5) govern the second area of data control, that is, the transfer of data from the input buses to the A- and B-sides of the adder. ROSDR(69–73) provides signals to control transfer from SBA to SAA. When no control is present, SAA(0–7) is, in effect, 0. Micro-orders allow true-complement transfer, crossgating, excess-6 adding for decimal operations, forcing of certain bits, and sign insertion (Diagram 4-401). ROSDR(74–77), (21–24, 81), (12–16), and (2–5) provide similar control for transfer from SBB to SAB. ROSDR bit 81 blocks ingating to SBB from ST, and allows gating ROSDR bits 21–24 into SBB as an emit field. If no bits are present in ROSDR (21–24), ROSDR(81) can be set to allow gating byte 1 or byte 2 of the N-register to SBB per ROSDR bits 2–5. A list of the micro-orders generated by

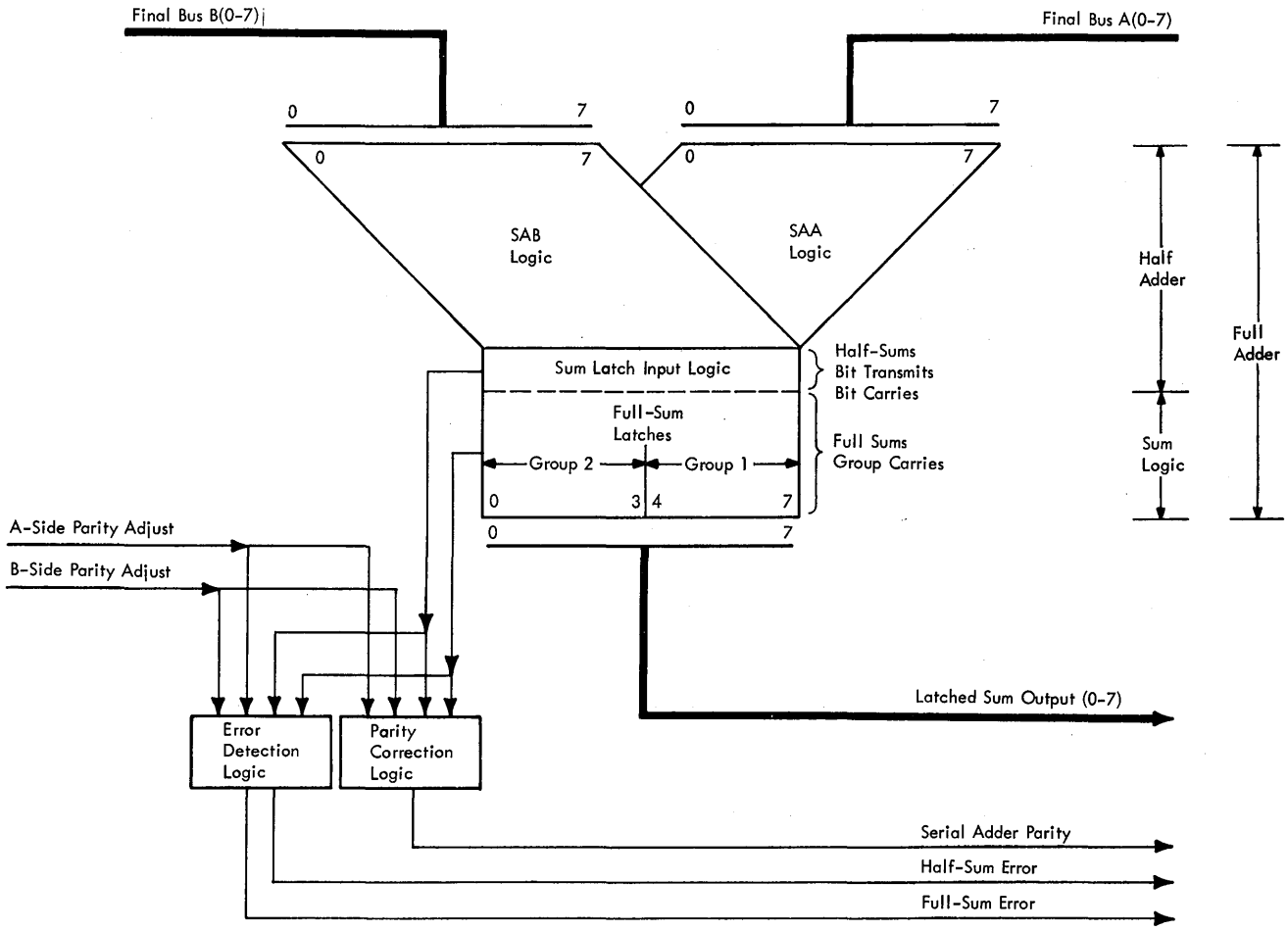
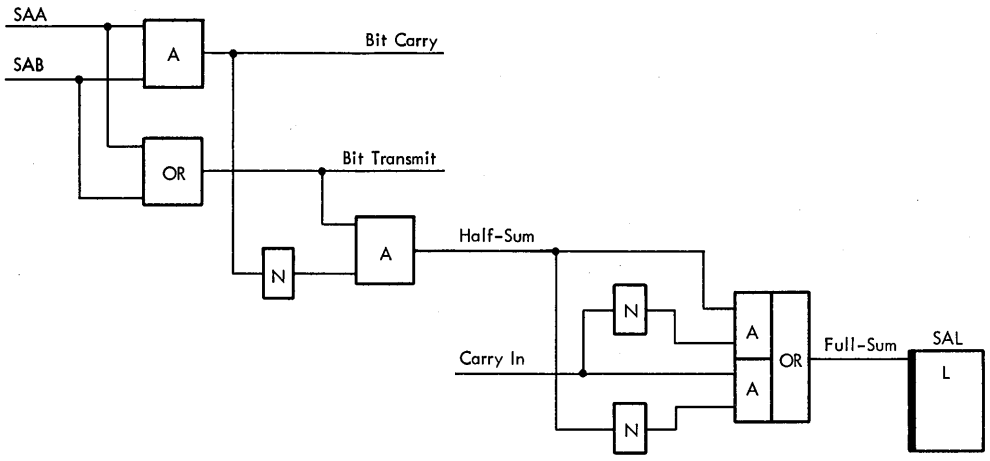


Figure 2-44. Serial Adder (Simplified)



SAA	0	1	0	1	0	1	0	1
SAB	0	0	1	1	0	0	1	1
Bit Carry	0	0	0	1	0	0	0	1
Bit Transmit	0	1	1	1	0	1	1	1
Half-Sum	0	1	1	0	0	1	1	0
Carry In	0	0	0	0	1	1	1	1
Full-Sum	0	1	1	0	1	0	0	1

Figure 2-45. Half-Sum and Full-Sum Logic

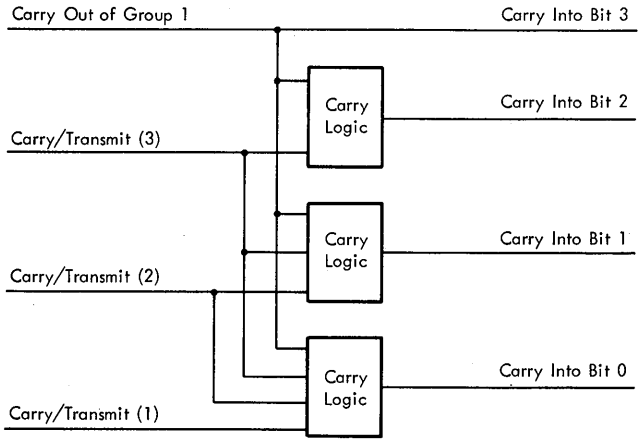


Figure 2-46. Carry Lookahead, Block Diagram

control fields M and N is contained in ALD A6261; by control fields C and E in ALD A6201, and control field W in A6261.

The third control area, SAL input, is also governed by ROS fields M and N. Controls include logical functions, decimal correction, product/quotient operations, and a hot-carry to SAL(3) or SAL(7) for complement add operations.

Functional Discription

Because the serial adder is used in many operations and is so versatile, a general discussion is not sufficient. Accordingly, the following paragraphs discuss the adder functions individually.

Binary Add

For binary add, data on SBA is entered in true or complement form and is combined with SBB data which may be 0's, forced bits, or data from ST or N. Combination takes place in the half-sum and full-sum logic, with carry signals from the carry lookahead logic (Diagram 4-402, FEMDM).

Decimal Operation

- Excess-6 is provided in input logic to SAA.
- Decimal correction is made in set-SAL logic.
- Validity tests are made on input digits and signs.

The excess-6 operation for decimal instructions is implemented by logical circuits rather than by using extra adder

cycles. The decimal character entering on SBA is increased by 6 as it is transferred to SAA (Diagram 4-403, FEMDM). Note that no time is lost in this operation; the circuits select the SAA positions which are 6 (0110) greater than the value on SBA.

The two operands are combined in the half-sum logic. If no group carry results, decimal correction is initiated by a ROS micro-order (Diagram 4-404, FEMDM). Decimal correction removes the excess-6 factor by using logical circuits to set SAL to a value 6 less than the full-sum value. Table 2-2 shows the decimal-corrected values for all possible erroneous characters. Again, because the circuits have been preconditioned, no cycles are lost, and the decimal operation proceeds at full speed.

Table 2-2. Decimal Correction for Erroneous Numeric Characters

Group 1 Result					Group 1 Result Decimal-Corrected (Binary Position)			
Decimal	Binary Position 4	5	6	7	4	5	6	7
15*	1	1	1	1	1	0	0	1
14*	1	1	1	0	1	0	0	0
13**	1	1	0	1	0	1	1	1
12**	1	1	0	0	0	1	1	0
11***	1	0	1	1	0	1	0	1
10***	1	0	1	0	0	1	0	0
9****	1	0	0	1	0	0	1	1
8****	1	0	0	0	0	0	1	0
7*	0	1	1	1	0	0	0	1
6*	0	1	1	0	0	0	0	0
5 ↕ 0	Valid digits; no correction required							

* 1-bits in positions 5 and 6: reset positions 5 and 6.
** A 0 in position 6: set position 6 and reset position 4.
*** A 1-bit in position 6 and a 0 in position 5: set position 5 and reset position 4.
**** 0's in positions 5 and 6; set position 6.

The following is an example of decimal correction for SAL(0–3) using Diagram 4-404:

Uncorrected binary result to SAL(0–3) = 1011.
SAL(3): No correction is made. It is set to 1 per binary addition.
SAL(2): Requires carry plus half-sum, or no carry plus no half-sum (effective 0 result). Conditions are not met and SAL(2) is not set.
SAL(1): Requires effective 0 and full-sum (2). Conditions are met and SAL(1) is set.
SAL(0): Requires effective 1 plus full-sum (1 and 2). Conditions are not met and SAL(0) is not set.
Corrected result in SAL(0–3) = 0101.

Incoming data is examined for validity on decimal instructions. If either character on SBA or SBB exceeds a value of 9 (binary 1001), an 'invalid digit' signal is generated and STAT E is set (Diagram 4-405, FEMDM). At the same time, 1's are forced into SAL to yield correct parity for the number transferred to S.

The invalid digit logic is also used to test the sign character entering the serial adder. An 'invalid sign' signal is developed if the sign character does not have a value of 10–15 (binary 1010–1111).

For multiply operations, the product is sent from B(66,67) to selected pairs of SAL bits to accumulate a byte of data. On non-decimal divide operations, bits are sent from A(4) and A(28) to selected SAL positions to accumulate a byte of data.

Logical Functions

Logical functions (AND, OR, and Exclusive-OR) are performed in the serial adder. These functions produce full-sum latch settings (carry information from adjacent positions is disregarded), as follows (Diagram 4-406, FEMDM):

1. AND: The combination of an active AND control signal and a bit-carry from the half-adder of that position.
2. OR: The combination of an active OR control signal with a bit-carry or a half-sum (in effect, a bit on either or both inputs) from the half-adder of that position.
3. Exclusive-OR. The combination of an active OER control signal with a half-sum (in effect, either input bit but not both) from the half-adder of that position.

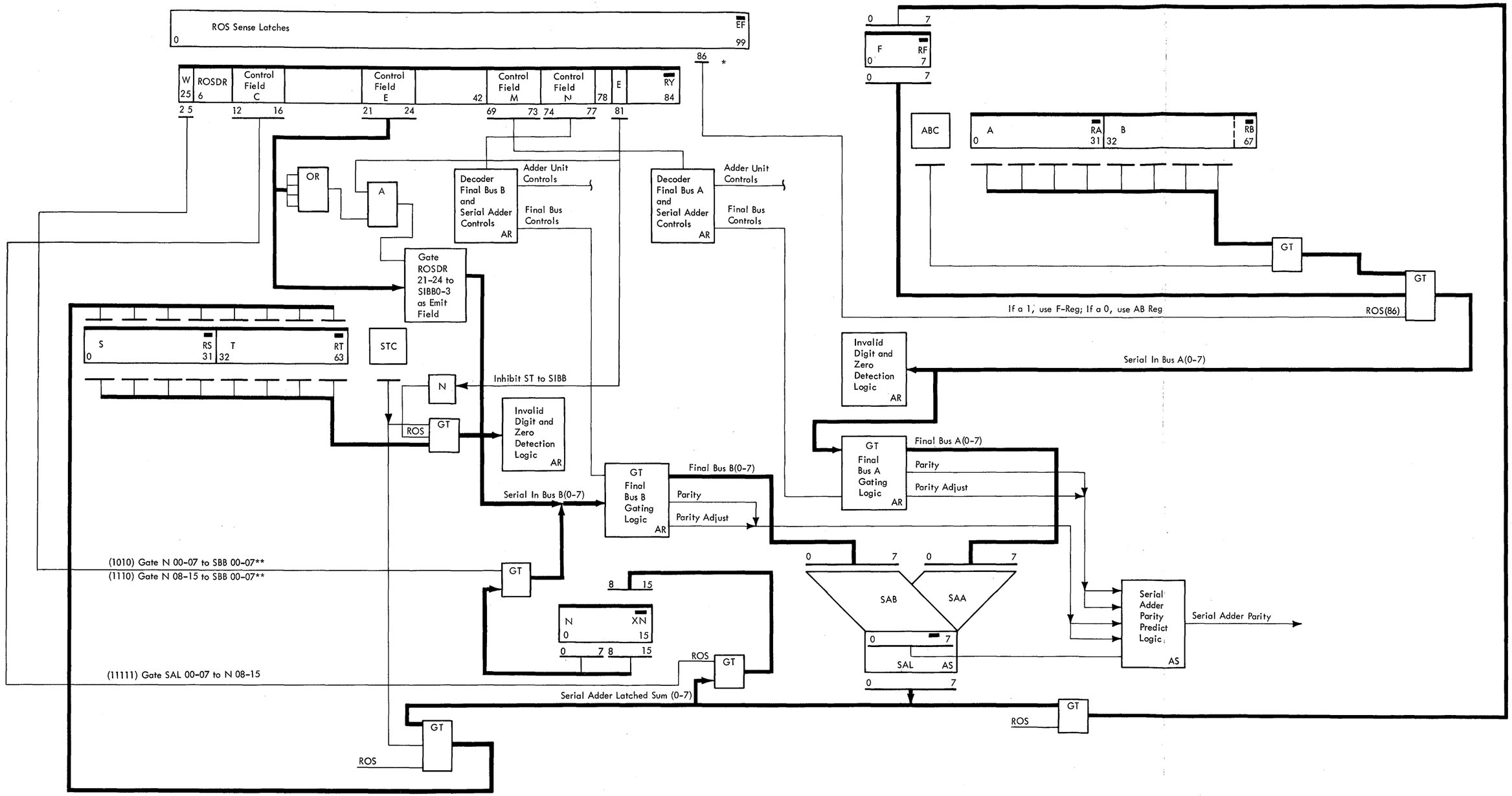
Results are transferred to selected bytes in ST or N.

Parity Correction

- Parity bit is set if numbers of bits in SAL(0–3) and SAL(4–7) are both odd or both even.
- Additional logic predicts parity for decimal operations.
- Parity is reversed on multiply and divide operations if only one bit is sent to SAL.
- Logical operations develop parity through unique logic.

Correct (odd) parity for the serial adder outputs is generated in the parity predict logic (Diagram 4-407, FEMDM). There are two basic areas of parity generation: (1) arithmetic and (2) logical (AND, OR, and exclusive-OR). The arithmetic parity generation is further divided into binary and decimal. See Figure 2-48, a block diagram of parity predict logic.

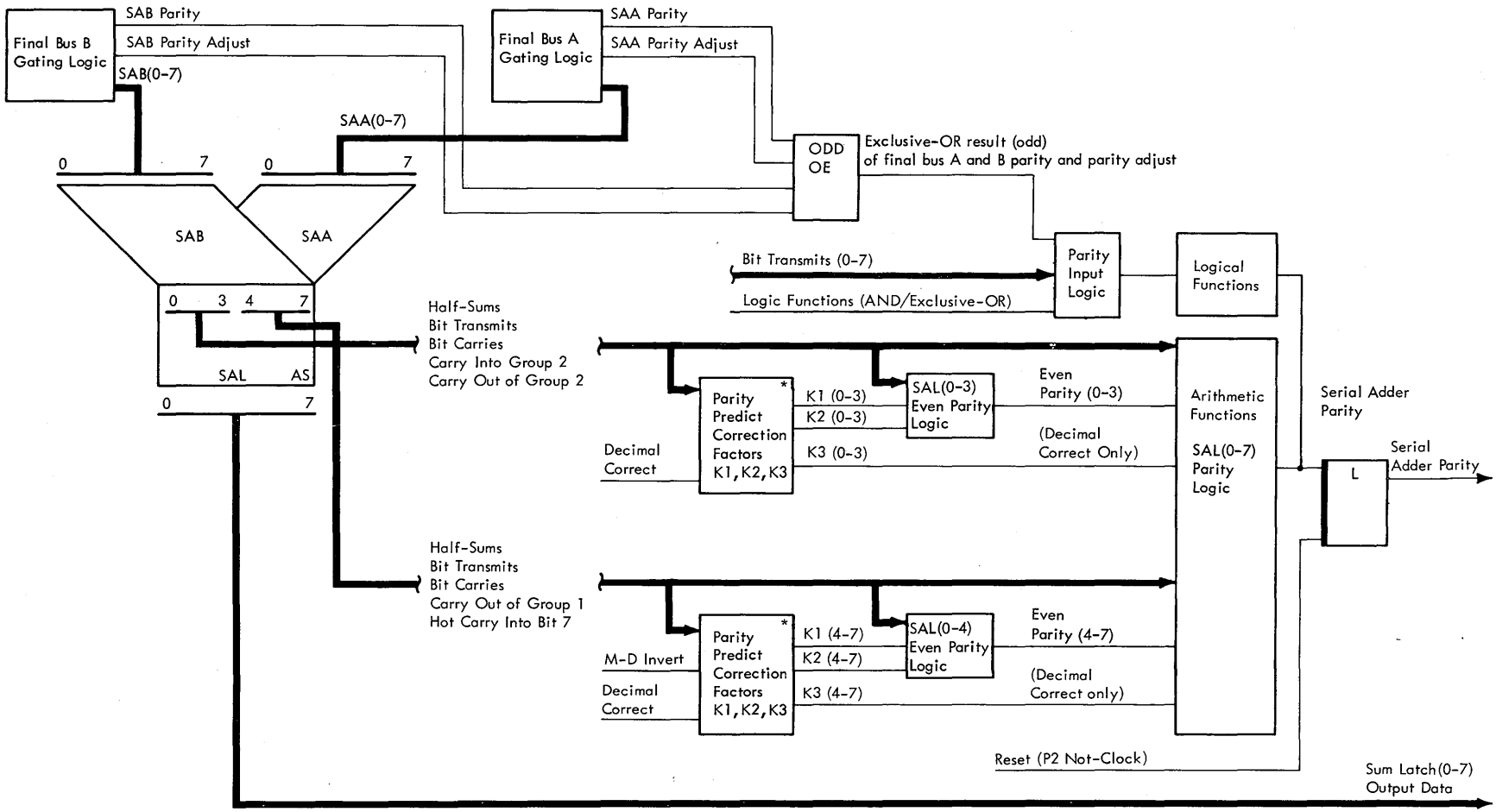
Binary parity predict logic includes factors K1 and K2, half-sum (0), and an odd or even number of transmit bits (1–3 and 5–7). K1 and K2 are established by carry and



* ROS control field R (bit 86) determines whether AB or F data is presented to final-bus-A gating logic.

**Either of these two micro-orders activates E16 micro-order (ROSDR 81) to block gating ST bytes to SiBB.

Figure 2-47. Serial Adder Gating Controls



*K1, K2, and K3 are defined factors of serial adder half-sum, bit-transmit, bit-carry, group-carry, and carry-into-group functions. (K3 factor is activated during decimal-correct operations only.)

Figure 2-48. Serial Adder Parity Predict Logic

transmit bits (Diagram 4-407). The signal resulting from these factors reflects the odd or even numbers of bits in SAL(0–3) and in SAL(4–7). If both are odd or both are even, the serial adder parity latch is set. If only one is odd, the parity latch is not set.

The shaded area in Diagram 4-407 indicates the additional control (K3) used for decimal operations. A decimal correction must be set up (decimal operation and no group carry) to allow energizing of the K3 signal. An examination of carry and half-sum (1, 2, 5, 6) allows the prediction of parity after the excess-6 factor is subtracted from SAL.

Multiply-divide operations present a different problem in that data is presented directly to SAL and is not processed through normal parity-predict logic. A test is performed on the two partial product bits or the one quotient bit (Diagram 4-408, FEMDM). A 1-bit change causes an 'invert predicted parity' signal which energizes K2 (4–7). Because no carries are generated, K2 would not normally be energized; thus, the K2 signal inverts the predicted parity.

Logical operations disable arithmetic parity prediction and energize a different-parity predict circuit. Three conditions are tested to set the SAL parity latch: (1) input parity, (2) parity adjust, and (3) odd-even transmit bits (Diagram 4-407). The development of each is as follows:

1. Parity: Normally presents the original parity (1 or 0) of the byte being entered on the input bus (one byte for SBA and one byte for SBB).
2. Parity-adjust: In effect, inverts the incoming parity regardless of the actual 1 or 0 parity. Parity-adjust is energized when a micro-order alters incoming data [for example, if SBA(0) enters as a 1 and a ROS micro-order forces SBA(0) to a 0, parity-adjust is energized].
3. Odd-even transmit bits: Exclusive-OR circuits analyze the developed transmit bits of all positions and produce signals denoting an odd or even number of transmit bits.

Parity generation for an OR function is based on the OR command and the OE transmit bits. Because only effective transmits are used to set SAL(0–7), parity generation needs only an even number of transmit bits to set the parity latch. The following is an example of parity generation for an OR function:

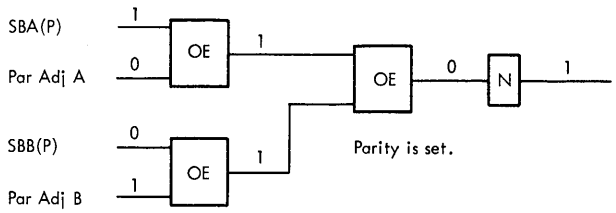
Bit	0	1	2	3	4	5	6	7	P	
SBA	1	0	1	0	0	0	1	0	0	
SBB	0	0	1	1	0	0	1	0	0	
Transmit	1	0	1	1	0	0	1	0		Even
SAL	1	0	1	1	0	0	1	0	1	

4 (even) transmit bits; Parity is set.

The exclusive-OR function requires the parity and parity-adjust exclusive-OR logic to generate correct parity. (Because the transmit bits do not reflect parity for exclusive-OR, they are not used.) Regardless of the number of 1 bits, if the parity of the two input data bytes is different, the resultant will be an odd number of bits, and the parity bit will not be set. An exception is caused by micro-order insertion of data over the byte data. This condition is corrected by means of the parity adjust circuits, as shown by the following example:

ROS Force 0

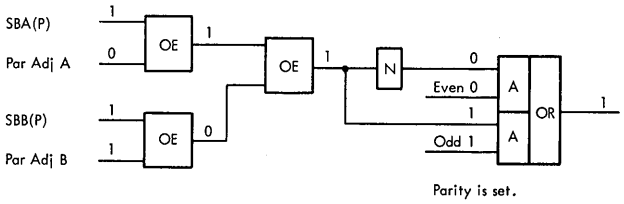
Bit	0	1	2	3	4	5	6	7	P
SBA	1	1	1	1	0	1	1	0	1
SBB	1	0	0	1	0	1	0	0	0
SAL	1	1	1	0	0	0	1	0	1



When the AND function is used, all three signals (SBA and SBB parity, SBA and SBB parity adjust, and odd-even transmit bits) are analyzed to generate parity. The serial adder parity latch is set when the result of the parity and parity-adjust signals matches the odd-even transmit bits signal, indicating an even number of bits have been generated; a parity bit is thus needed. An example of parity generation for a logical AND operation follows:

ROS Force 0

Bit	0	1	2	3	4	5	6	7	P	
SBA	0	0	1	0	1	1	0	1	1	
SBB	1	1	1	1	1	0	1	0	1	
Transmit	0	1	1	1	1	1	1	1		Odd
SAL	0	0	1	0	1	0	0	0	1	



Note: In four special cases the SBB input parity bit is set: (1) no-operation, when SBA is all 0's, (2) when -64 (1100 0000 binary) is forced, (3) when subtract 1 (1111 1111 binary) is forced, and (4) on partial product entry when SBB is 0. In addition, the SBB parity bit is held off on add 1 (0000 0001 binary) because it is incorrect parity.

When an invalid digit is detected on a decimal operation and all 1's are forced to SAL(0–7), the parity predict logic is bypassed, and the SAL parity latch is set to give proper parity for the byte sent to ST.

Error Detection

Serial adder data is parity-checked on both a half-sum and a full-sum basis. Error indications are retained in the half-sum error or full-sum error latches (Figure 2-49).

Half-sum error logic tests the incoming data for accuracy. The signals tested are (1) half-sum(0–7), and (2) input parity and parity-adjust for both A- and B-sides of the adder. When an input parity is in error (even parity), the combined signals produce an odd result, and the 'half-sum error' trigger is set.

Full-sum error logic tests the accuracy of the final answer by combining the SAL outputs and the resulting state of the parity latch. An even result sets the 'full-sum error' trigger.

The error triggers remain set until reset by the 'error reset gate' signal. To avoid meaningless error indications and subsequent logout operations, the set error trigger signals are blocked when:

1. The 'inhibit serial adder parity check' micro-order is active.
2. An invalid digit is detected during a decimal operation (all serial adder latches, including the parity latch, are set).
3. The 'logout' trigger is set.

PARALLEL ADDER

The parallel adder, 60 data bits plus parity, performs arithmetic and logical functions and is involved in most intra-CE data transfers. Data flow for the parallel adder is shown in Figure 2-50.

The parallel adder bit positions are divided into sections and groups to implement carry lookahead and parity-predict functions (Figure 2-51). Additional functions illustrated in the figure are half-sum, full-sum, and latch-shifter logic.

Data Input

Data is transferred to the parallel adder from various registers by means of input buses controlled by ROS micro-orders (Figure 2-52). More than one bus may enter the same side of the parallel adder, but only one bus is active at a given time.

ROS fields T and U control inputs to bus B and bus A, respectively. Gate control triggers for adding B and T are shown in Diagram 4-409, FEMDM. ROS sense latches are decoded at P2 time to select a gate control trigger; the trigger remains set until the following P1 time to make the data available as long as the parallel adder needs it. Note in Diagram 4-409 that only three of the four bits of the ROS fields are used; the state of the fourth bit does not affect the gate control triggers shown but does affect other gate control triggers.

Individual Bit-Position Logic

- Full-binary capabilities (half-adder and full-sum logic) are provided.

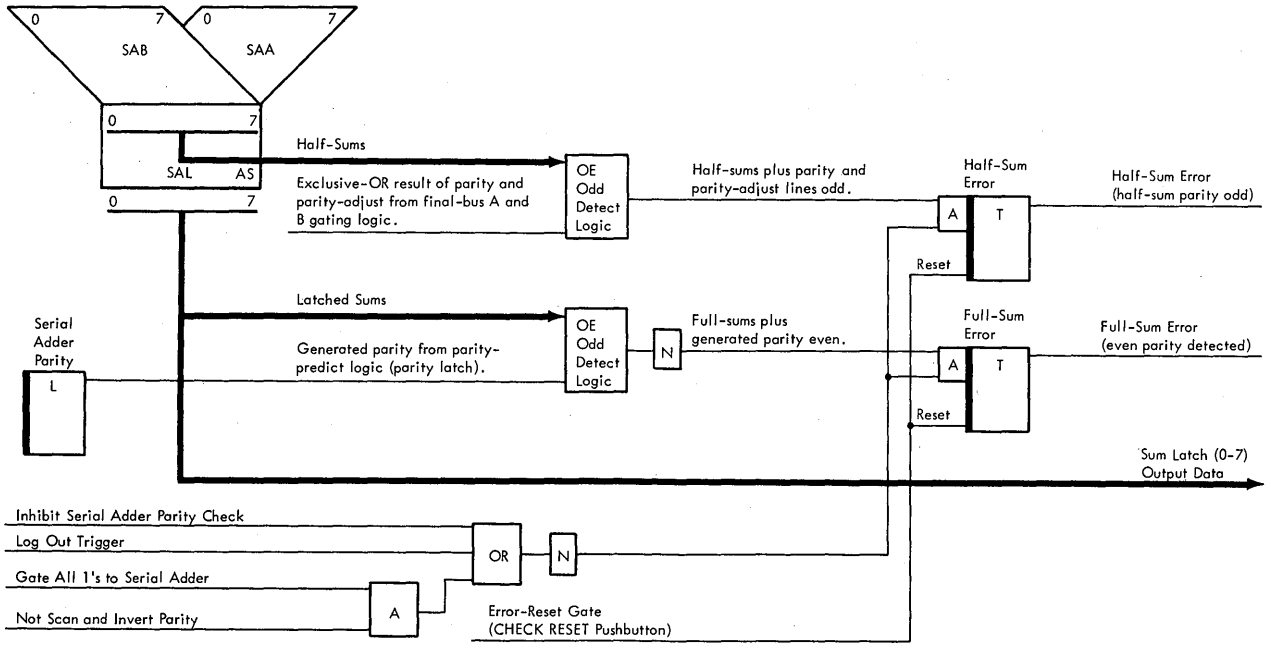


Figure 2-49. Half-Sum and Full-Sum Error Logic

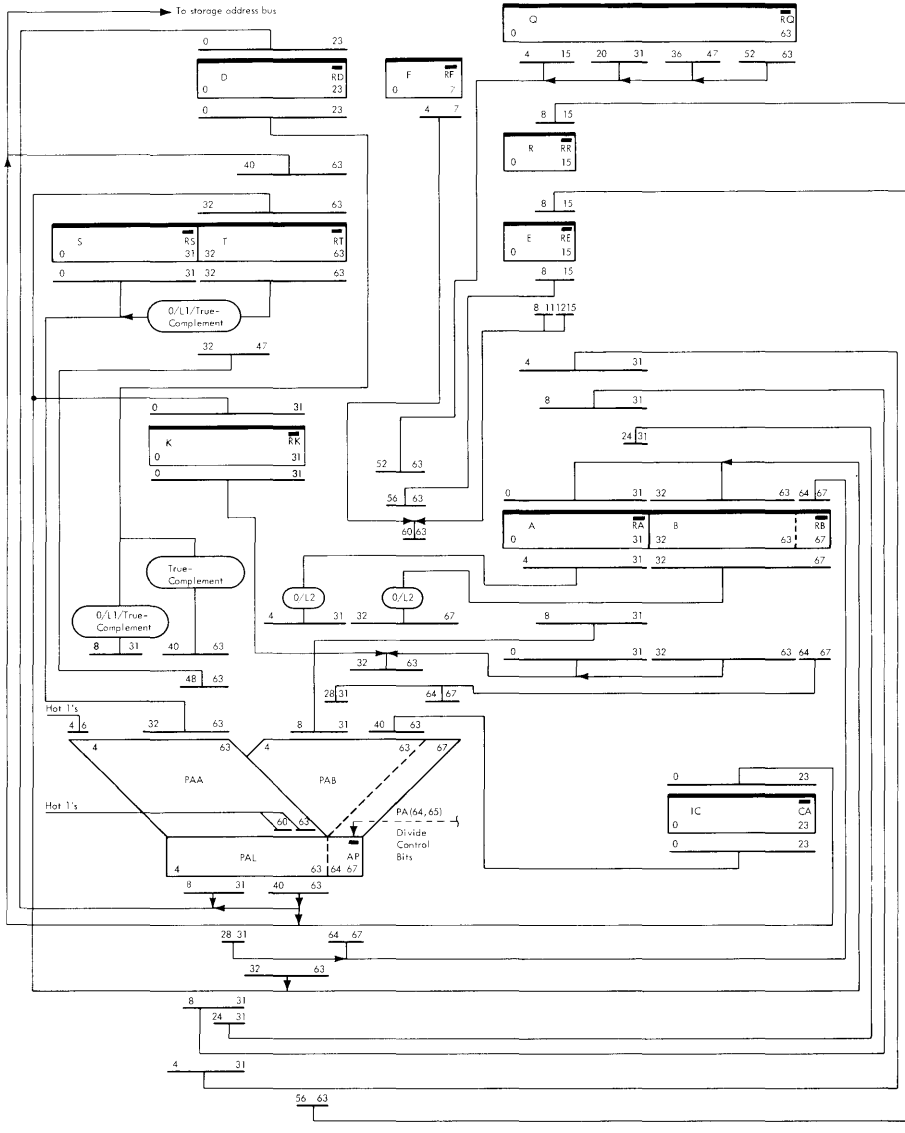


Figure 2-50. Parallel Adder Data Flow

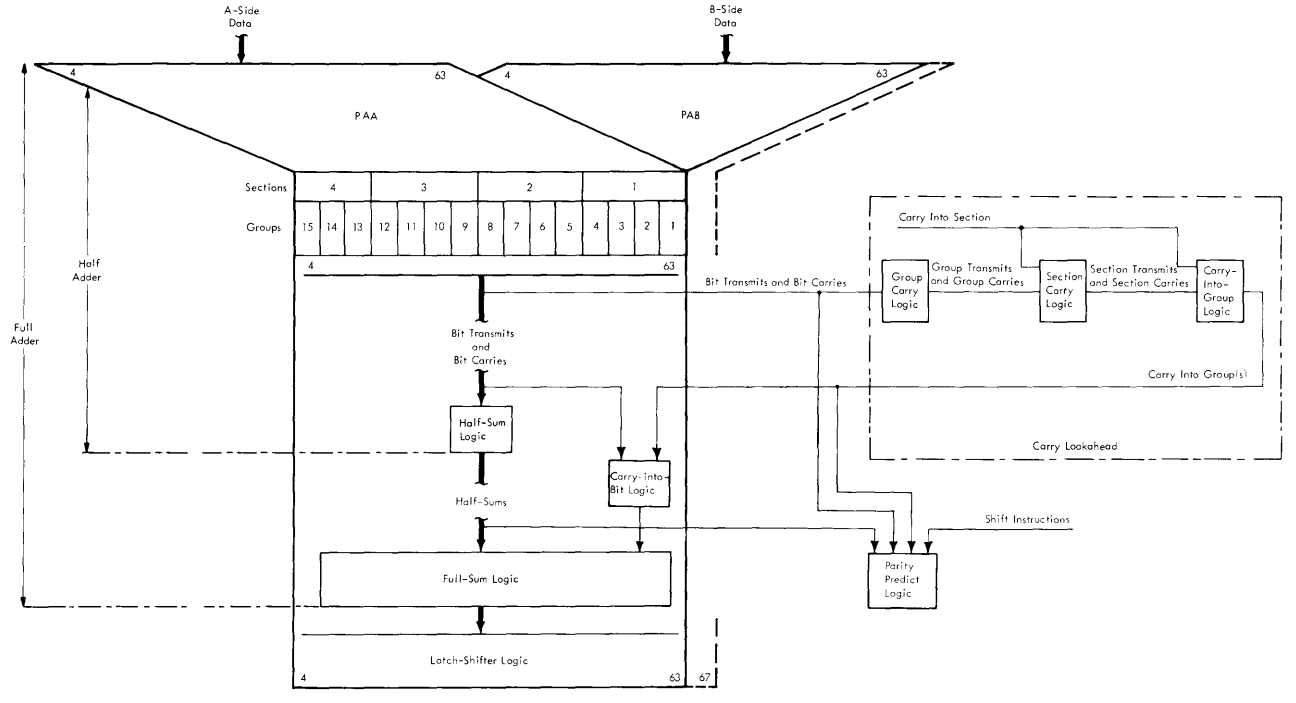


Figure 2-51. Parallel Adder Function Breakdown

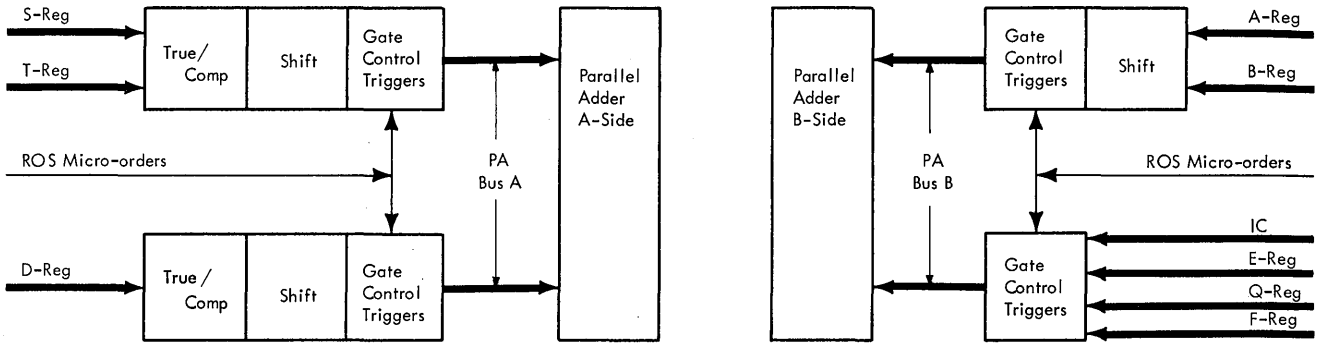


Figure 2-52. Parallel Adder Input Buses

- Shift logic (latch-shifter) is included at the output.

The logic functions associated with each adder position are shown in block form in Figure 2-53. These functions (half-adder, carry-into-bit, full-sum, and latch-shifter) constitute the full-binary logic of each bit position; operation is as follows. The status of corresponding A- and B-side operand bits is entered into the half-adder, where they are combined to produce bit-transmits, bit-carries, and half-sums. The bit-transmit/bit-carry is sent to carry lookahead logic to produce predicted carry information, and the half-sum is sent to the full-sum logic. The carry-into-bit logic combines the immediately available bit-transmit/bit-carry from adjacent lower-order adder positions (representing an actual carry) with the somewhat later-returning predicted carry. This predicted-carry output (carry-into-bit) is also sent to the full-sum logic, where it is combined with the half-sum from the half-adder logic to generate a final full sum (1 or 0) for that adder position. This full sum or, possibly, the full sum from four positions to either the left or the right (depending on the particular shift control) is gated into the adder latch. Latched sum data is retained until the following cycle for sampling into the selected register(s). The following paragraphs discuss the logic involved in each function.

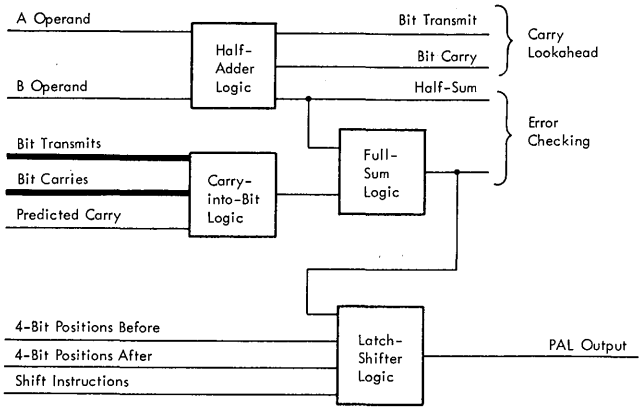


Figure 2-53. Bit Position Block Diagram

Half Adder

The logic involved in bit-transmit, bit-carry, and half-sum functions is:

1. Bit-transmit: At least one and possibly two 1-bits are contained in the two corresponding A- and B-side operand positions.

   Note: For certain operations, the parallel adder is set to all 1's by a micro-order which forces all 1's into the A-side of the half-adder (Diagram 4-410, FEMDM).

2. Bit-carry: Two 1-bits are contained in the corresponding A- and B-side operand positions.
   Half-sum: A single 1-bit, but not two, is contained in the corresponding A- and B-side operand positions.

Carry-into-Bit Logic

- Detects "carry-into" conditions affecting each particular bit position.

The carry-into-bit logic of each adder position detects whether a carry-into condition prevails, resulting from either an actual carry or a predicted carry. Carry-into-bit circuitry logically ORs the actual carry (prevailing carry conditions from immediately adjacent lower-order positions) with predicted carry (carry-into-group indications from lookahead logic, signifying that effective carry conditions exist in the more extreme lower-order areas).

The actual carry into any particular adder position is determined by logically testing all remaining lower-order bit positions within that same group or, if the particular adder position happens to be the low-order group position, testing

all four bit positions of the next lower-order group. (Predicted carries are discussed in "Carry Lookahead".)

Note: Either an actual carry (from adjacent positions) or a predicted carry (from carry lookahead) is allowed to affect a particular position, but not both. Where both carries occur, conditions producing the actual carry also function to inhibit the predicted carry from entering the affected bit position(s). This inhibit logic is illustrated in Diagram 4-410, FEMDM, as follows: Carry conditions from positions 48–51 generate both a predicted carry to position 47 (carry-into-group 5), via carry lookahead logic, and an actual carry in the form of bit-transmit/bit-carry signals. Because group 4 positions (48–51) represent the actual carry source, the group-4-carry condition is then inverted to inhibit predicted carry entries into position 47.

Full-Sum Logic

The full-sum logic for any particular adder position combines (by means of an exclusive-OR) the carry-into-bit output with the half-sum output of the half-adder, developing a 1 or 0 full-sum for that adder position.

Latch-Shifter Logic

Latch-shifter logic facilitates the left 4/right 4 shifting of the full sum during the same cycle in which it is developed. (Logical and data-transfer operations also utilize this logic.) For any particular adder position, zero-shift, left-4 shift, and right-4 shift controls (respectively) gate the full sum into the latch associated with that position, into the latch associated with the position four places to the left, and into the latch associated with the position four places to the right. (Scan-out operations also utilize the latch-shifter logic but only for its data path facilities.) All latches retain the latched sum until the following cycle, and extended-clock signals delay resetting the latches long enough for the error-checking logic to function.

Note: An adder-hold ('–→HOLD') micro-order, used during certain operations, blocks the extended-clock reset signal and causes the latches to retain their data for one additional cycle.

Carry Lookahead

- Predicts carry before full-sum development.
- Reduces time required to provide full sum.
- Lookahead logic divides 60-position adder into 15 four-bit groups and divides these groups into four sections.
- Lookahead information is developed in form of bit-position carry, group carry, and section carry and is then fed back into individual positions as predicted carries.

The carry lookahead function provides the adder with the capability of entering full sums directly into the adder latches. Lookahead functions effectively predict the carry resulting from combining two operands and use this predicted carry to convert half-sums to full sums before the entry of information into the adder latches. This sequence eliminates the additional time required by ripple operations, which would be necessary in converting half-sums to full sums if half-sums were entered directly into the latches.

For design reasons, the 60-position adder is divided into 15 four-position groups; these groups are subdivided into four sections. This group/section arrangement reduces the logic decoding required in implementing the carry lookahead functions. Group/section arrangement and carry lookahead data flow are shown in Figure 2-54.

Lookahead logic is designed so that, for any particular position, the effective carry conditions in all lower-order positions (except for an adjacent few) are logically predicted for that position. Carry conditions that would later be produced in these same adjacent few positions as a result of propagated lower-order carries are predetermined by the lookahead functions and logically entered into that position as a predicted carry. Two things must be known to predict a carry for a particular bit position: (1) whether carries exist in any of the lower-order bit positions and (2) whether intervening bit positions can transmit a carry to the bit position in question. This information is available to the carry lookahead logic in terms of bit-carries and bit-transmits. Using this method, each bit position then requires only that logic necessary to detect prevailing carries (actual carry) in the adjacent positions and to logically OR the actual and predicted carries when developing the full sum. Predicted carries are presented to the input logic of individual positions as 'carry into group' signals. Figure 2-55 illustrates the adder areas supplying source information for actual and predicted carry signals to adder position 44. Note that, although lower-order carry conditions exist in both examples, they are represented by an actual carry in example 1 and by a predicted carry (carry-into-group-5) in example 2. (Recall also, from the previous discussions, that where both actual and predicted carries are generated to a particular position, only the actual carry is entered; the predicted carry entry is blocked.)

In the lookahead logic, predicted carry information is developed by testing each adder group for bit-position carry

conditions, combining these conditions to form group-carry conditions, and then similarly combining the group-carry indications to produce section-carry conditions. All lower-order carry conditions affecting any individual adder position (with the exception of the positions immediately adjacent to that position) are then collectively represented to the lookahead logic as section-level carry information. Section-carry information from each section is then (after being combined with lower-order section-carry indications) sent to higher-order sections, where it is combined with the group-carry conditions within these sections to produce 'carry into group' signals. As previously described, these 'carry into group' (predicted-carry) signals are then logically ORed with actual carries within the carry-into-bit logic of each individual adder position and combined with half-sum information to generate a full sum. (Recall also that, where both actual and predicted carries are generated to a particular position, only the actual carry is entered and the predicted carry entry is blocked.)

The following paragraphs give detailed descriptions of group-level and section-level carry-predict functions.

Group-Level Carry Logic

- Bit-position carry conditions are combined in four-bit groups to generate group-carry conditions.
- Group-carry logic outputs define effective status of all bit positions composing a group.
- Group-carry outputs are sent to section-carry logic.

The group-level carry information generated for any particular group is determined by logically combining the bit-transmit/bit-carry outputs of all four positions within that group. Group-transmit/group-carry signals are then generated and sent to the section-level carry logic of the section in which the particular group is located. Group-level carry logic outputs indicate the following:

1. Group transmit: Signifies that all bit positions within that group have received at least one bit of operand data; i.e., bit-transmit conditions exist throughout the group.
2. Group carry: Signifies that bit-transmit/bit-carry conditions within that group are such that an effective carry condition exists from the high-order position of that group.

Group-level carry logic is illustrated in Diagram 4-411, FEMDM. Note (in the diagram) that group-transmit/group-carry outputs are determined solely by bit-transmit/bit-carry conditions, which represent incoming data only (without the use of any propagated carry information).

When group-carry conditions are sent to their associated section-level carry logic, they may also (at the same time) generate 'carry into group' signals to adjacent higher-order groups within that same section. This sequence results in the immediate propagation of group-carry information (within that same section). Carry-into-group circuits that are not activated at this particular time may be activated somewhat later by incoming section-carry signals from lower-order sections.

Because group-carry conditions are used in developing section-carry conditions, a time differential exists between the two logic functions. The timing relationships between bit-carry, group-carry, section-carry, and carry-into-group are discussed in "Arithmetic Function Sequence".

Section-Level Carry Logic

- Group-level carry conditions are combined to develop section-level carry conditions.
- Section-carry outputs define effective carry conditions of all bit positions within a particular section.
- Section-carry outputs are sent to higher-order sections as predicted carry information.

Section-level carry information is determined by logically combining the group-transmit/group-carry outputs of all groups within a section. Like group-carry generation, section-carry outputs are also determined solely by group-carry conditions (without the use of any carry propagation). Section-level carry logic outputs signify the following:

1. Section-transmit: Indicates that all bit positions of all groups within that section have received at least one bit of operand data (i.e., group-transmit conditions exist throughout the section).
2. Section-carry: Signifies that group-transmit/group-carry conditions within that section indicate that an effective carry condition exists in the high-order bit position of that section.

The section-level carry logic (Diagram 4-411) develops a 'section 1 transmit' signal from group transmits and a 'section 1 carry' signal from combinations of group-transmits and carries.

Section-Level Carry-Into Logic

A section-carry generates a carry into the next-higher-order section. The section-level-carry-into logic (Diagram 4-411)

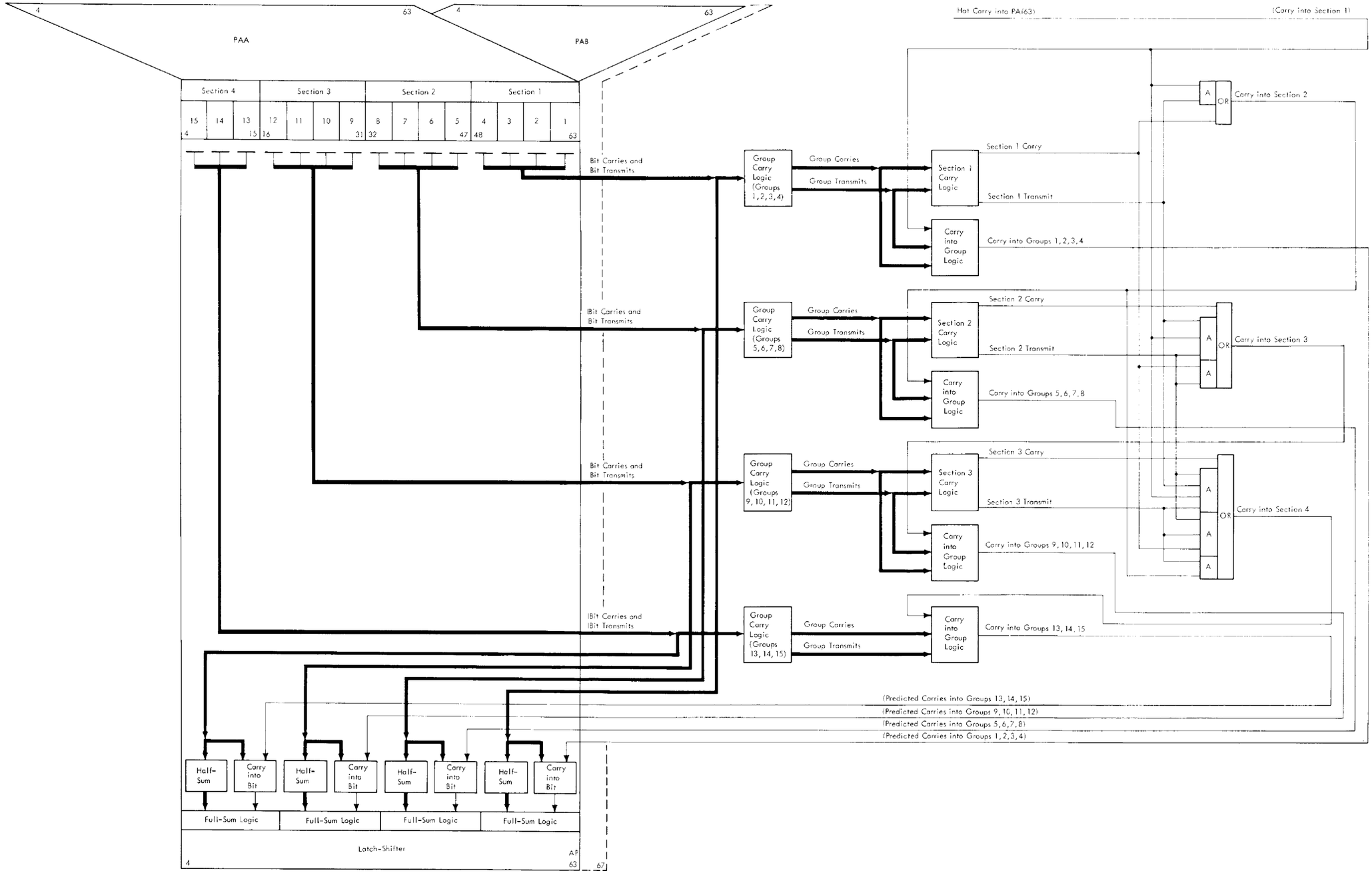


Figure 2-54. Parallel Adder Carry Lookahead Data Flow

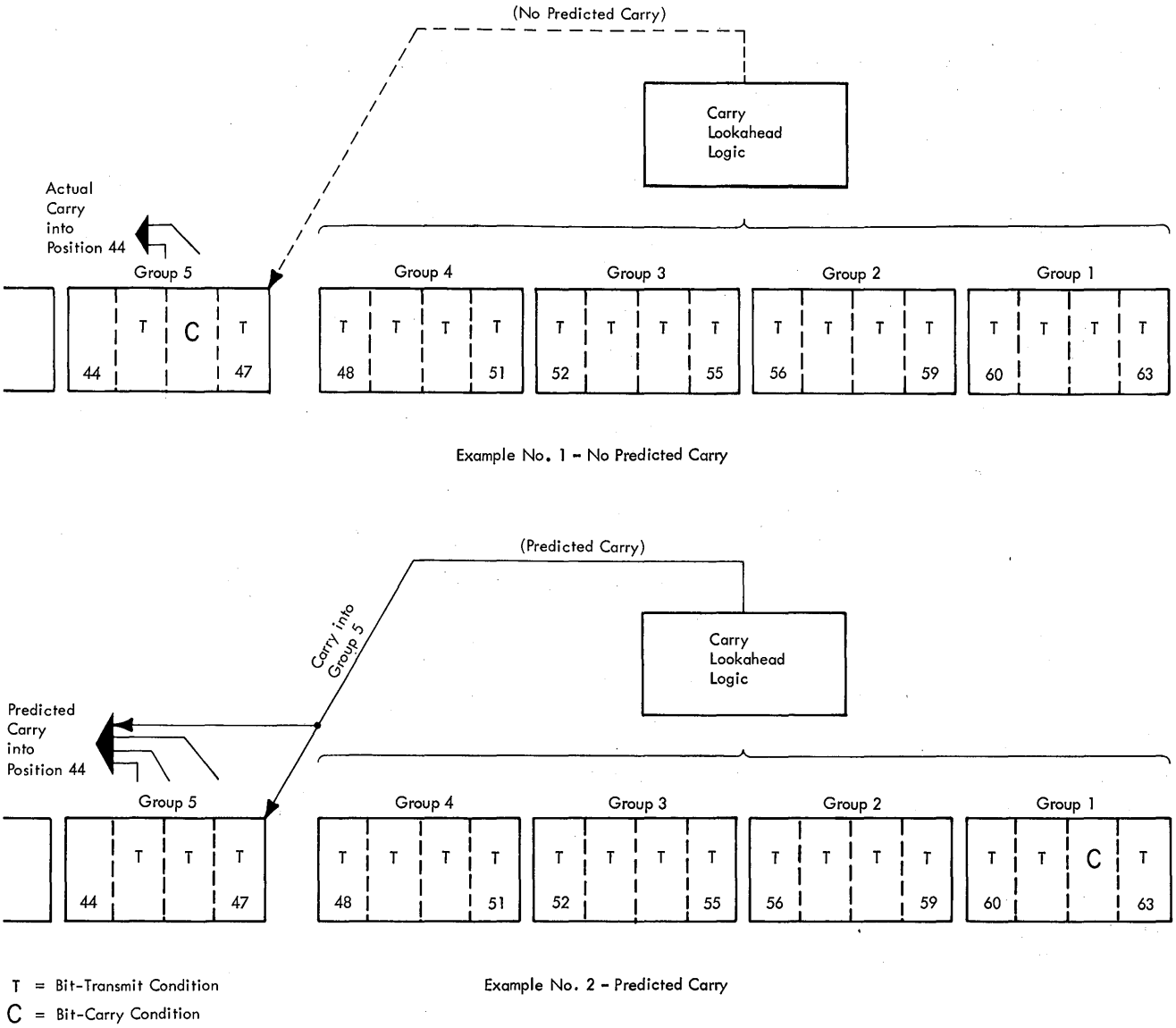


Figure 2-55. Actual and Predicted Carry Origin for PA(44)

develops carry-into-section signals, starting with a 'carry into section 1' signal produced by a hot-carry. Carry into sections 2, 3, and 4 are developed by section-transmit and section-carry logic.

Group-Level Carry-Into Logic

Carry-into-section signals produce a carry-into-group signal for at least the low-order group of the section. Development of additional carry-into-group signals is dependent on group carry/transmit conditions (Diagram 4-411). Note that carry-into-group signals may be developed independently from the carry-into-section signals.

Bit-Level Carry-Into Logic

Carry lookahead conditions the bit-level carry-into logic of the low-order group position if no group carry is present from the next-lower-order group. Other bits in the group are conditioned if intervening low-order transmit bits are present (Diagram 4-411). For example, if a bit 49 carry and a bit 48 transmit have been developed, the result is a group 4 carry that generates a section 1 carry. The section 1 carry and a section 2 transmit produce a 'carry into section 3' signal. A 'carry into group 9' signal results from the 'carry into section 3' signal. When bit-transmits 30 and 31 are present, a carry into bits 29 and 30 takes place to develop full sums.

Diagram 4-411 shows the timing relationships for carry lookahead. Note that although a direct carry occurs before a carry lookahead, this time difference does not affect the final sum development which takes place after all carry circuits have settled down.

Full-Sum Development

- Half-sums are combined with carry information (actual and predicted) to develop full sum.

The manner in which carry lookahead and half-sum functions are logically combined to produce a full-sum result is illustrated in Figure 2-56. Note that all group-level and section-level functions are arranged on a section (four-section) basis, whereas carry-into-bit and full-sum functions appear in each adder position.

The complete carry lookahead system is shown in Diagram 4-411; a summary of carry-predict operation is as follows. When operand data is presented to the A- and B-sides of the adder, the half-adders of all positions are sampled for bit-transmit/bit-carry information. (Half-sums are also generated from the half-adders and presented to the full-sum logic of each position at this time.) All bit-transmit/bit-carry information is sent to the associated group-carry logic, and all group-carry outputs are entered into their respective section-carry logic. Section-carry outputs now represent the carry status that logically prevails in the high-order position of each section (without any effects of carry propagation). All section-level carry outputs are then combined with lower-order section-carry information to determine whether a 'carry into section' (predicted-carry) signal is generated for the higher-order section(s). 'Carry into section' signals sent to higher-order sections combine with group-carry conditions within those sections to produce the carry-into-group conditions that represent predicted carries for the individual bit positions. The carry-into-bit logic of each individual position then logically ORs 'carry into group' (predicted-carry) signals with actual carry indications, and this output combines with the half-sum to produce the full-sum result.

Note that throughout the lookahead sequence no ripple operations are required. Definite cycle times, however, are associated with each predict function (bit-carry, group-carry, section-carry, carry-into-section, and carry-into-group); these times are discussed in the following paragraph.

Arithmetic Function Sequence

- Eight logical delay levels are required for arithmetic functions.

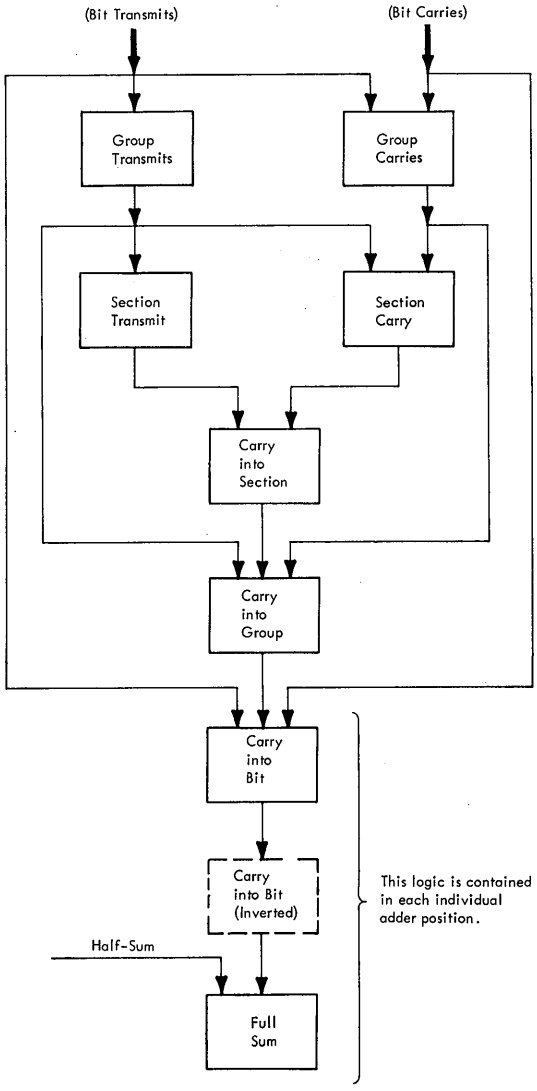


Figure 2-56. Full-Sum Development Logic

- Extended clock signals are used within adder.
- Full-sum results are latched (retained) for 1 cycle.

The timing sequence in which all adder logic operates to develop and check full-sum information is shown in Figure 2-57. Three delay levels (P4–P7) occur between the time at which data is placed on the adder input bus (by the associated gate-control triggers) and the time at which the same data enters the half-adders. (Two of these delay levels result from bus-gating delays; the third, from the signal cables.) Eight levels of signal delay, therefore, are required

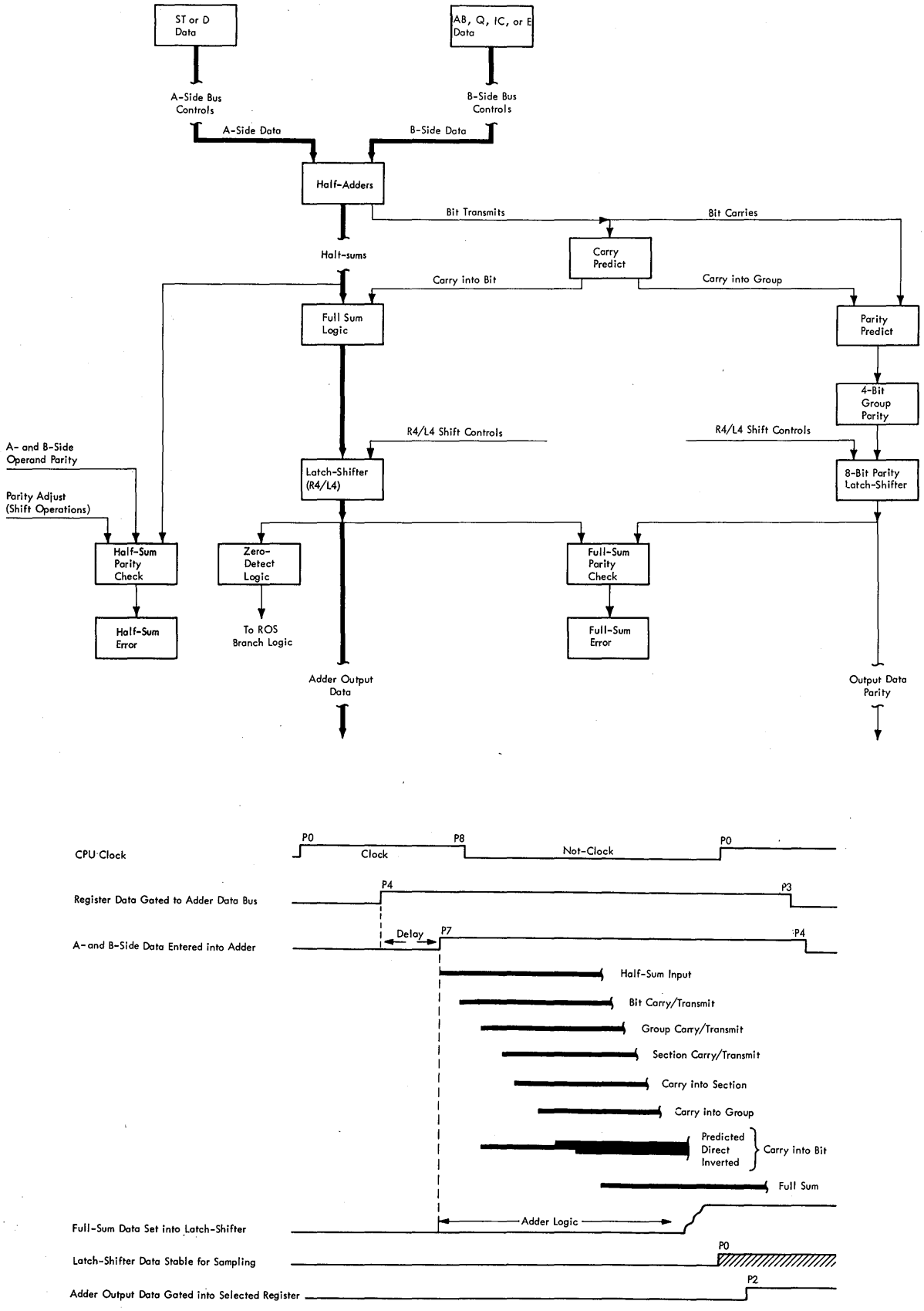


Figure 2-57. Parallel Adder Logic Function Sequence

within the adder for the full-sum development process. As noted on the timing chart in Figure 2-57, the logic functions that require the eight delay times occur in the following sequence:

1. Bit-carry/transmit.
2. Group-carry/transmit; carry-into-bit (direct).
3. Section-carry/transmit.
4. Carry-into-section.
5. Carry-into-group.
6. Carry-into-bit (predicted).
7. Carry-into-bit (inverted).
8. Full-sum.

Note: A carry-into-bit can originate early from a direct carry or late from the predicted carry logic.

Extended clock signals are used within the parallel adder to control all latches. The clock portion of the normal CPU clock signal is extended two delay levels (approximately 20 ns), producing a symmetrical clock signal of 100-ns clock and not-clock times. These extended clock signals result in delaying both the setting and resetting of the adder latches. Delaying the setting of the full-sum latches provides additional time for carry-predict functions, and delaying the resetting of the latches retains latched sum information long enough for sampling by the error-checking logic.

Full-sum information contained in the adder latches is normally retained one cycle. For certain operations, however, an adder-hold ('→HOLD') micro-order inhibits the clock signal that resets the adder latches, thus retaining latched sum information for one additional cycle.

Parity-Predict Logic

- Odd parity is supplied with each byte (or half-byte) of adder output data.
- Parity generation is simultaneous with full-sum development.
- Parity-predict logic utilizes inputs from half-adders and carry lookahead logic.
- Parity generation is corrected accordingly for left-4/right-4 data shifting.

Odd parity is generated for each byte (or half-byte in the case of positions 4–7 and 64–67). Predict logic is employed, allowing parity information to be generated simultaneously with the development of full-sum data. (This scheme eliminates the time involved in analyzing the full-sum bit count to determine parity.) Parity is initially predicted for each four-bit group of adder output data. For the eight-bit byte outputs, the parity information predicted for the two adjacent four-bit groups that constitute a particular byte is combined (exclusive-ORed) to determine the full-sum parity of that byte. Because the adder is also capable of shifting full-sum data left 4 and right 4 (before entry into the adder latches), generation of correct byte parity for left-4/right-4 operations then becomes a matter of selecting which two adjacent four-bit group parity outputs to combine when determining the parity of a particular output byte.

Parity is logically predicted through functions of the incoming operand data; operation is as follows. At the same time half-adder outputs are sent to the lookahead logic (to predict carry information), they are also sent to four-bit group parity-predict logic. A typical four-bit group parity-predict function is shown in Diagram 4-412, A, FEMDM. (Group 4 is used as an example; all groups are similar.) For each four-bit group, bit-transmit, bit-carry, and half-sum outputs from half adders and carry-into-group outputs from the lookahead logic are combined to logically predict whether the resultant full-sum bit count for that particular group will be odd or even. Note in the diagram that duplicate decoder logic is present in each four-bit group parity-predict circuit. This duplicate logic simultaneously produces the opposite polarity (out-of-phase) signals required for use in the eight-bit parity latch-shifter logic without the signal delay introduced if an additional inversion component were used.

Typical parity latch-shifter logic used in combining two adjacent four-bit group parities to determine eight-bit byte parity is shown in Diagram 4-412, B. (Adder output byte 48–55 is used as an example; all byte parity logic is similar.) Note in the diagram that the two four-bit group parity outputs to be combined (Exclusive-ORed) when determining byte parity are selected according to the type of shift operation in process, i.e., left 4, right 4, or no shift (straight transfer). The generated parity for each adder byte (or half-byte in the case of positions 4–7 and 64–67) is set into the corresponding parity latches for transfer with the data and sent to the full-sum error-checking logic. Because parity information is used in full-sum error checking and both parity and full-sum information are formed independently, an inconsistency in either will cause a full-sum error.

Error Checking

Parallel adder logic employs both half-sum and full-sum checking facilities. Half-sum checking verifies incoming data (in regard to assigned parity only); this test also results in verifying half-adder operations because half-sum outputs are used in half-sum checking logic. Full-sum checking logic compares the full-sum bit count (odd/even) with the

generated parity information on a byte (or half-byte) basis. Because full-sum and parity information are formed independently, an inconsistency in either results in a full-sum error.

Half-Sum Checking

- Compares half-sums with incoming operand parity.

Half-sum checking logic combines the parity information assigned to incoming A- and B-side operand data with the half-sum generated when the same two operands are combined in the half-adders. This combining of parity and half-sums is performed on a byte (or half-byte) basis, with detected errors stopping the CE clock and lighting indicators signifying the byte (or half-byte) in error.

Half-sum checking logic, illustrated in Diagram 4-413, FEMDM, operates as follows. A stage of precheck logic for each byte (or half-byte) combines half-sums with the corresponding A- and B-side parity information in odd-detect (exclusive-OR) circuits. (The precheck logic shown in Diagram 4-413 monitors adder positions 48–55.) This logic functions so that, if the number of half-sums plus the A- and B-side parity bits, results in an odd bit count, the half-sum precheck trigger for that associated adder area is set. Precheck outputs from all adder areas are then combined with left-shift logic at the input to the 'half-sum error' trigger. This left-shift logic determines whether an actual half-sum error exists or whether shifting the register data left 1 or left 2 positions (while en route to the adder) has forced a half-sum error.

Note: Left-shifting the adder input data left 1 or left 2 positions invalidates the assigned parity information, thus forcing half-sum errors. The number of half-sum errors created, however, should result in an even number; i.e., the half-sum errors forced into positions 4–31 should equal the number of half-sum errors forced into positions 32–63. Odd-detect logic, therefore, allows only an odd number of half-sum precheck indications to set the 'half-sum error' trigger during a left 1/left 2 shift operation.

If a valid half-sum error exists, the 'half-sum error' trigger is set, thus setting the 'final error' latch. Setting the 'final error' latch prevents the 'half-sum error' trigger from automatically resetting, which in turn prevents resetting the precheck logic for the area in which the half-sum error occurred. Inhibiting these resets causes the CE program to stop on the following cycle (provided the CE CHECK CONTROL switch is in the STOP position), with the HALF SUM error indicator for the area incurring the error displayed on the roller switch indicators.

The half-sum error indications are reset by the 'error reset gate' signal (SYSTEM RESET or CHECK RESET pushbutton).

Full-Sum Checking

- Compares latched sum information with generated parity.

Full-sum checking logic combines latched sum information with generated parity information on a byte (or half-byte) basis. (Full-sum checking for adder positions 48–55 is shown in Diagram 4-414, FEMDM.) Because the CE operates with odd parity, combining full-sum bits with generated parity should always result in an odd bit count. Detecting an even latched-sum-plus-parity bit count sets the 'full-sum error' trigger for that particular adder byte area, which, in turn, sets the 'final error' latch. The error signal that sets the 'full-sum error' trigger stops the CE program on the following cycle (provided the CE CHECK CONTROL switch is in the STOP position) and lights a FULL-SUM error indicator on the roller switch indicators, signifying the area incurring the full-sum error.

For practical reasons, combining full sums with parity is logically accomplished by first exclusive-ORing the generated parity with a single latched sum position (Diagram 4-414) and, then, combining that result with the remaining latched sums of that particular byte. An odd result (signifying an even overall bit count) then sets the associated 'full-sum error' trigger.

Full-sum error conditions are reset by the 'error reset gate' signal (SYSTEM RESET or CHECK RESET pushbutton).

Convert-to-Decimal Operation

Special circuits, used only in the convert-to-decimal operation, provide excess-6 decimal correction when required. Excess-6 is forced on the PAB bus when a test of a four-bit group indicates a decimal value higher than 9. For this operation, parallel adder bit positions 28–63 are logically divided into four bit groups, each group representing a decimal digit in the packed format. Diagram 4-415, FEMDM, shows the development of excess-6 signals for PAB(28–31) and PAB(60–63). Note that these signals are activated only when ROS has developed the excess-6 gate and when the AB bits indicate the need for decimal correction.

For this operation, data is brought into the parallel adder one bit at a time by transferring one byte of data to the serial adder and sampling SAL(0). If SAL(0) = 1, the

'conv dec' trigger is set and a hot-carry sets PAL(63) (Figure 2-58). The contents of the serial adder are then shifted left 1 position so that the next bit can be sent to the parallel adder (which is also shifted left one position). [For details, see Chapter 3, Section 2, "Convert to Decimal, CVD (4E)".] Because data is processed through normal parallel adder entry logic, parity generation takes place in a normal manner.

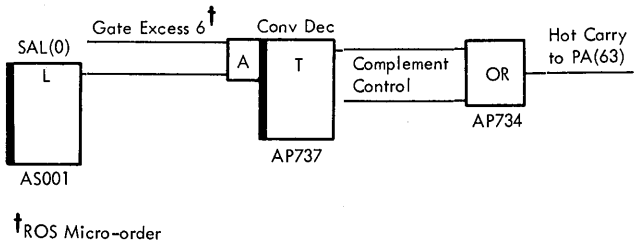


Figure 2-58. Convert-to-Decimal Data Flow to Parallel Adder

Set Condition Code

After an operation, PAL is analyzed to set the PSW condition code (CC). CCs are set in many ways, with many variables for different instructions; Diagram 4-416, FEMDM, shows a typical example. Various sections of PAL are examined for a 0 condition; combinations of PAL equal 0 and micro-orders set STAT A, which is sampled by the instruction and the result sign to set the CC.

In the example, if PAL(32–63) is not equal to 0 and the result is negative, a CC of 1 is set on a fixed-point operation. (This setting indicates a number less than 0.) Note that an overflow condition on a fixed-point instruction sets both CC bits, regardless of the condition of PAL.

For a floating-point operation PAL(7–67) is examined for 0. A not-0 condition and a plus result set a CC of 2.

SECTION 6. STATUS AND CONTROL TRIGGERS

This section discusses the eight status triggers (STATs A–H) and miscellaneous control triggers. A summary of the conditions that set STAT's A–H is shown in Figure 2-59, and a typical example of STAT logic (showing STAT B) is illustrated in Diagram 4-501, FEMDM.

STAT A

- STAT A indicates:
  Zero condition for parallel adder.
  Nonzero condition for serial adder.
  Digit condition on edit operations.

STAT A primarily indicates zero-detect conditions. Except during scan-in, when it is set directly to the value of T(54), STAT A is normally set at P2 clock time by one of the following conditions:

1. 'Set STAA if SAL(0–7) not equal to zero' signal, with SAL(0–7) not containing all 0's.
2. 'Edit set STAA' signal (edit operations).
3. 'Serial adder (0–3 or 4–7) not zero' signal, from the 'serial adder not zero' (SNZ) latch (indicating that the serial adder latched outputs do not contain all 0's).

   Note: The 'SNZ' latch is set at not-clock time by the 'decimal correct 0–3 set STAT's AE', or 'serial carry-7 STAT's AE decimal correct 4–7' signal.

4. 'Set STAA if PAL(7–63) equals zero' signal, with PAL(7–63) latched outputs containing all 0's.
5. 'Set STAA if PAL(32–63) equals zero' signal, with PAL(32–63) latched outputs containing all 0's.
6. 'Set STAA if PAL equals zero and insert sign' signal, with E(6) = 1.

The output of STAT A is entered directly into a polarity-hold latch, which unconditionally assumes the same binary state as STAT A at P1 not-clock time. (This latch retains its assumed state until not-clock time of the cycle in which STAT A is reset.)

STAT A is reset at P1 clock time if one of the following conditions is active:

1. 'STAT trigger reset' signal (conditioned by either a 'system reset' or an 'I-fetch reset' signal).
2. 'Reset STAA' signal, which is conditioned when any one of the following is active:
   a. 'Edit reset STAA' signal.
   b. 'Set STAA if PAL(0–63) equals zero' signal.
   c. 'Set STAA if PAL(32–63) equals zero' and signal.
   d. 'Set STAA if PAL(32–63) equals zero and insert sign' signal, with E(6) = 1.
   e. 'Reset STAA if PAL(32–63) not equal zero' signal, with STAA polarity-hold latch set.

      Note: If PAL(32–63) contains all 0's and the STAA polarity-hold latch is set, the 'reset STAA if PAL(32–63) not equal zero' signal is inhibited from resetting STAT A.

   f. 'Set STAA if SAL(0–7) not equal zero' signal.

STAT B

- STAT B indicates:
  Zero-condition for serial adder.
  Overflow condition for decimal, fixed point, left-shift operations.
  Condition of PAL(31).
  Condition of B(32).

STAT B primarily indicates overflow conditions. Except during scan-in, when it is set directly to the value of T(55), STAT B is normally set at P2 clock time by one of the following conditions:

1. 'Set STAB if SAL(0–7) equals zero' signal, with SAL(0–7) containing all 0's.
2. 'Set STAB on decimal overflow' signal, with the 'decimal overflow' latch set.

   Note: The 'decimal overflow' latch is set at not-clock time of a decimal-compare cycle in which:
   a. The 'serial adder in bus A(7)' contains a 1-bit, and either STAT A, STAT D, or STAT H is reset.
   b. The 'serial adder in bus A(0–6)' is not equal to 0.
   c. STAT H is set, with STAT C and STAT F either both set or reset.

3. 'Set STAB if PAL(31) equals 1' signal, with PAL(31) = 1.
4. 'Gate fixed-point overflow to STAB' signal, with a fixed-point overflow condition prevailing.
5. 'Set STAB on left shift overflow' signal, with a left-shift overflow condition detected.
6. 'B(32) to STAB and T(32) to STAG' signal, with B(32) = 1. [STAT B is set at P2 + 140 ns under this condition to allow B(32) to become stable before it is sampled.]

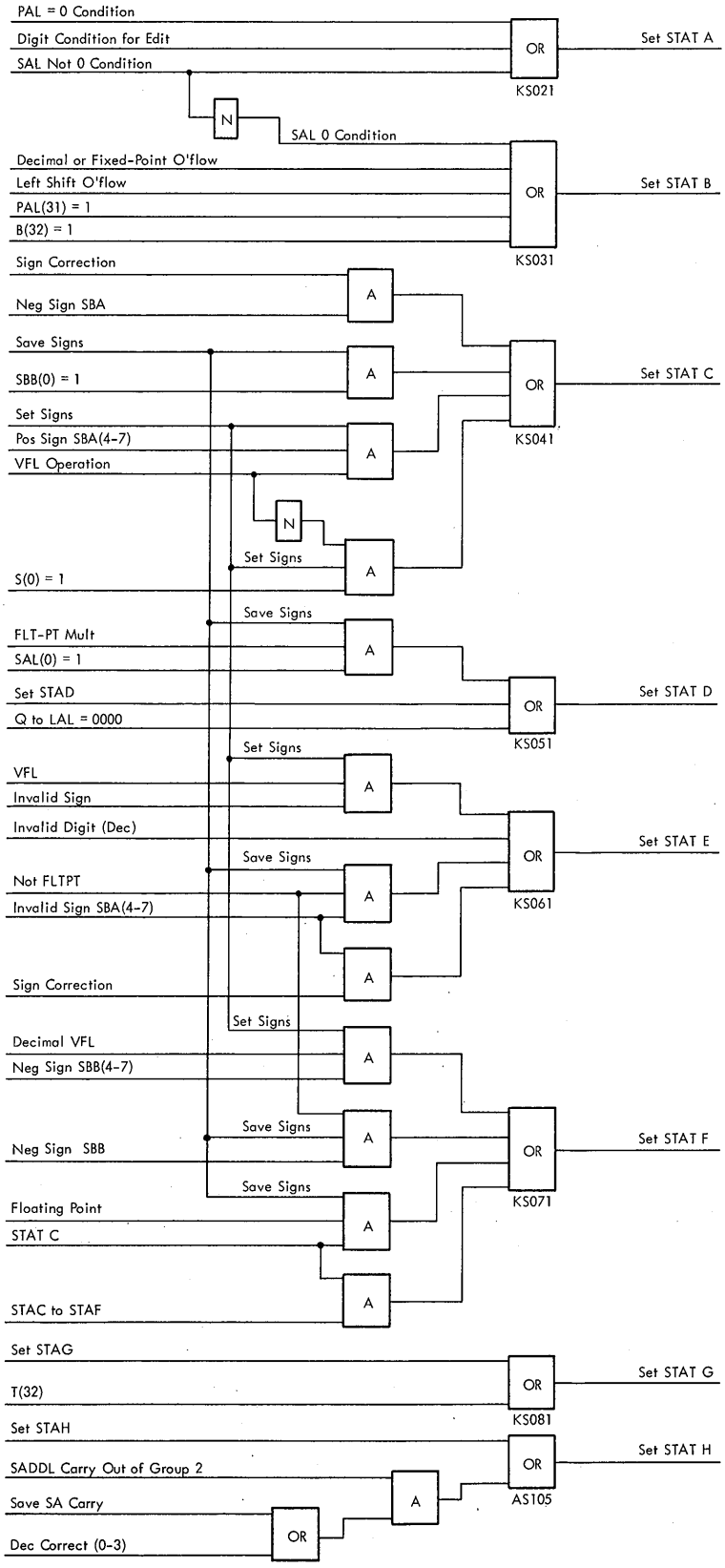


Figure 2-59. Summary of Setting of STAT's

The outputs of STAT B are sent directly to a polarity-hold latch. This latch unconditionally assumes the state of STAT B at not-clock time of the cycle in which STAT B is set and retains this information until not-clock time of the cycle in which STAT B is reset. The output from the STAT B polarity-hold latch inhibits the resetting of STAT B whenever STAT B is set during the same cycle in which either a fixed-point overflow or a left-shift overflow is detected. (Either of these overflow conditions causes a program interruption requiring that STAT B remain set for interrogation.)

STAT B is normally reset at P1 clock time if one of the following conditions is active:

1. 'STAT reset' signal (conditioned by either a 'system reset' or an 'I-fetch reset' signal).
2. 'Reset STAB' signal, which is conditioned if one of the following signals is active:
   a. 'Set STAB if PAL(31) equals 1'.
   b. 'Gate fixed-point overflow to STAB'.
   c. 'Set STAB on left shift overflow'.
   d. 'B(32) to STAB and T(32) to STAG' (resets STAT B at P1 + 140 ns).

STAT C

- STAT C holds:
  'Serial adder in bus A' sign on sign-correction VFL operations.
  'Serial adder in bus B' sign on save-signs VFL operations.
  Sign for set-signs on VFL and non-VFL operations.

STAT C primarily indicates the sign of a source operand. Except during scan-in, when it is set to the value of T(56), STAT C is set at P2 + 140 ns clock time by one of the following conditions:

1. 'Sign correct SA(4–7)' signal, with a negative sign detected on 'serial adder in bus A'.
2. 'Save signs' signal, with 'serial adder in bus B' position 0 containing a 1-bit during subtract or compare operations and a 0-bit during all others.
3. 'Set signs' signal during VFL instructions in which 'serial adder in bus A' positions 4–7 contain a positive sign during subtract or compare operations and a negative sign during all others.
4. 'Set signs' signal during any non-VFL operation in which S(0) = 1.
5. 'SATR set STAC' signal.

STAT C is normally reset at P1 clock time by the 'STAT trigger reset' signal (activated by either a 'system reset' or an 'I-fetch reset' signal). STAT C is also reset at P1 + 140 ns whenever the 'set signs' signal is activated for operations other than VFL operations or when activated by 'SATR reset STAC' signal.

STAT D

- Used by microprogram to retain ROS branch information.
- Indicates sign for save-signs operation on floating-point multiply and divide.
- Indicates Q-to-LAL equals 0000.

STAT D stores a characteristic carry from SAL(0) during floating-point multiply and divide operations and indicates that the B1 or B2 field of an instruction equals 0. For operations other than floating-point multiply and divide and scan-in operations, STAT D is available for arbitrary microprogram use and can be unconditionally set or reset by the 'set STAT D' and 'reset STAT D' signals, respectively. (Such operations include storing of the dividend sign on fixed-point operations.) Except during scan-in operations, when it is set directly to the value of T(57), STAT D is normally set at P2 + 140 ns by one of the following conditions:

1. 'Set STAT D' signal.
2. 'Save signs' signal during floating-point multiply and divide operations in which SAL(0) = 1. (STAT D is set at P2 clock time under this condition.)
3. 'Gate Q to LAL 0000' signal.

   Note: This signal is activated whenever the B1 or B2 field of an instruction is being gated from Q to LAL and is found to equal 0. Although STAT D is always set on this condition, its significance is of value only during an SS-format instruction when a B2 = 0000 indication must be retained for more than one cycle. (Used in setting ROSAR when selecting I-fetch ROS words for SS-format instructions.)

4. 'Set D on set or insert key' signal.

STAT D is normally reset at P1 clock time by the 'STAT trigger reset' signal, which is activated by either a 'system reset' or an 'I-fetch reset' signal, 'Reset STAD' and 'gate I-fetch invalid address' signals reset STAT D at P0 + 140 ns. The 'reset STAD on decimal overflow' signal resets STAT D at P2 clock time.

STAT E

- Indicates invalid digits and signs.

STAT E primarily indicates the detection of invalid data during decimal operations. Except during scan-in, when it is set to the value of T(58), STAT E is normally set at clock P2 + 140 ns by one of the following conditions:

1. 'Set signs' signal during VFL operations in which an invalid sign is detected on either the 'serial adder in bus-A' or '-B'.
2. 'Save signs' signal for operations other than floating-point operations in which an invalid sign is detected on the 'serial adder in bus B(4–7)'.
3. 'Sign correct SA(4–7)' signal with the detection of an invalid sign on the 'serial adder in bus A(4–7)'.
4. Detection of an invalid digit on either side of the 'serial adder in bus'.
5. Detection of an invalid digit on the 'serial adder in bus A(0–3)', with the 'digit examine' latch set (edit operations).

STAT E is reset at P1 clock time by the 'STAT trigger reset' signal (activated by either a 'system reset' or an 'I-fetch reset' signal).

6. 'Set STAE I/O error' signal.

STAT F

- STAT F holds:
  'Serial adder in bus B' sign on set-signs decimal operations.
  'Serial adder in bus B' sign on save-signs operations.
  Condition of STAT C.

STAT F primarily indicates the sign of VFL destination operands. Except during scan-in, when it is set to the value of T(59), STAT F is normally set at P2 + 140 ns by one of the following conditions:

1. 'Set signs' signal during a VFL decimal operation, with a negative sign detected on 'serial adder in bus B(4–7)'.
2. 'Save signs' signal during operations other than floating-point operations, with a negative sign detected on 'serial adder in bus B'.
3. 'Save signs' signal during a floating-point operation, with A(0) = 1.
4. 'STAC to STAF' signal, with STAT C set. (Sets STAT F at P2 clock time.)

STAT F is normally reset at P1 clock time by the 'STAT trigger reset' signal (activated by a 'system reset' or an 'I-fetch reset' signal). The 'STAC to STAF' signal also resets STAT F at P0 clock time in preparation for setting STAT F again at P2.

STAT G

- Used by microprogram to retain ROS branching information.

- Indicates state of T(32).

STAT G is available for arbitrary microprogram use and for indicating the state of T(32). Except during scan-in, when it is set to the value of T(60), STAT G is normally set at P2 + 140 ns by one of the following conditions:

1. 'B(32) to STAB and T(32) to STAG' signal, with T(32) = 1.
2. 'Set STAG' signal.

STAT G is normally reset at P0 + 140 ns by the 'reset STAG' or 'B(32) to STAB and T(32) to STAG' signal. A 'system reset' signal and an 'I-fetch reset' signal also reset STAT G.

STAT H

- Used for serial adder carry-control functions and ROS branching information.

STAT H indicates a serial-adder carry. Except during scan-in, when it is set to the value of T(38), STAT H is set by one of the following conditions:

1. 'Set STAH' signal and clock time P2 + 140 ns.
2. The output of a latch set at not-clock time by either a 'decimal correct 0–3 and set STAT's AE' or a 'save serial adder carry' signal in conjunction with a serial adder carry from group 2. The set condition is timed at P2 clock time.

STAT H is reset at P0 clock time with the following signals: (1) 'branch on ATR select in', (2) 'dec cor 0–3 set STATS AE', (3) 'Ith I-fetch reset', and (4) 'save SA carry'. It is also reset at P2 clock time with 'reset serial carry to STAH' and by 'master reset' with no time consideration.

CONTROL TRIGGERS

A number of control triggers perform functions similar to STATs. Table 2-3 lists the most significant triggers, summarizes their functions, and provides ALD and FETOM references.

Table 2-3. Control Triggers

Trigger	Function	ALD Reference	FETOM Reference	Roller Switch Indicator
Right Digit	Selects digit from AB byte on edit operations.	KZ321	Volume 2, Chapter 3, Section 5, "General Data Handling"	RT DIG Roller 4 Position 4 Bit 32
S	Indicates source character, rather than fill character, on edit character transfers.	KZ321	Volume 2, Chapter 3, Section 5, "Introduction to Edit Operation"	S Roller 4 Position 4 Bit 33
Leave	Controls 'serial adder bus B' on edit operations.	KZ201		LEAVE Roller 4 Position 4 Bit 34
Step ABC	Increments ABC on edit operations, if 'right digit' trigger is set.	KZ501		STEP ABC Roller 4 Position 4 Bit 35
Block I-Fetch	Prevents most I-Fetch functions when interruption or exceptional condition is to be processed.	KD501	Volume 2, Chapter 3, Section 1, "Block I-Fetch Trigger"	BLOCK Roller 4 Position 5 Bit 8
Branch Invalid Address	Indicates branch address of successful branch is invalid.	KD701	Volume 2, Chapter 3, Section 1, "Invalid Address Detection"	BR INVLD ADR Roller 4 Position 5 Bit 16
I-Fetch Invalid Address	Indicates Q has been refilled from invalid address.	KD711	Volume 2, Chapter 3, Section 1, "Invalid Address Detection"	INVLD ADR Roller 4 Position 5 Bit 17
Instruction Length Not Available	Resets ILC in old PSW and resets all interrupt code triggers except 'interrupt code 4' trigger. Set by "late" storage protection check.	KM851	Volume 2, Chapter 3, Section 1, "Fetch Protection Detection"	IL NOT AVAIL Roller 4 Position 5 Bit 18
Time Clock at Limit	Indicates timer has been decremented past 0 and requests external interruption	KM221	Volume 2, Chapter 3, Section 1, "Timer Exceptional Condition"	TC AT LIMIT Roller 4 Position 5 Bit 19
Timing Gate	Controls duration of I/O control or direct-control signals. Set and reset by microprogram.	KX311	Volume 2, Chapter 3, Section 7, "Write Direct, WRD(84)" and "Read Direct, RDD(85)"	TIME GATE TGR Roller 4 Position 5 Bit 35

Table 2-3. Control Triggers (Cont.)

Trigger	Function	ALD Reference	FETOM Reference	Roller Switch Indicator
No Retry†	Indicates to diagnostic programmer instruction retry may give unpredictable results. Set by (1) 'store per D' signal, (2) 'PAL to IC' signal and not SS format, and (3) 'write local stor' signal and LSWR not selected.	KS321		NO RETRY Roller 4 Position 1 Bit 18
IC in LSWR†	Indicates IC is saved in LSWR. Occurs only on SS format operations.	KS321		IC IN LSWR Roller 4 Position 1 Bit 19

† 'No retry' and 'IC in LSWR' triggers perform no control function but indicate machine conditions only.

SECTION 7. STORAGE CONTROL INTERFACE

The storage control interface (SCI), an integral part of the Computing Element, governs the transfer of all information between the Computing Element and the Storage and Display Elements in the system. The SCI regulates the flow of addresses, data, key information, and control signals associated with main storage.

GENERAL DESCRIPTION

The CE uses a combination of simplex, distributed simplex, and multiple driver simplex lines to communicate with SEs and DEs. Figure 2-60 shows how these lines connect each CE, through its storage control interface (SCI), with all SEs and DEs in the system.

Basic Interface Considerations

The buses and control lines for the SCI are shown in Figure 2-61. The term "simplex" refers to the lines going from one CE to each SE and DE. A signal sent from a CE on a simplex line is available to only one SE or DE. The term "distributed simplex" refers to the lines going from one CE to all SEs and DEs in the system. A signal sent from a CE on a distributed simplex line is available to all SEs and DEs.

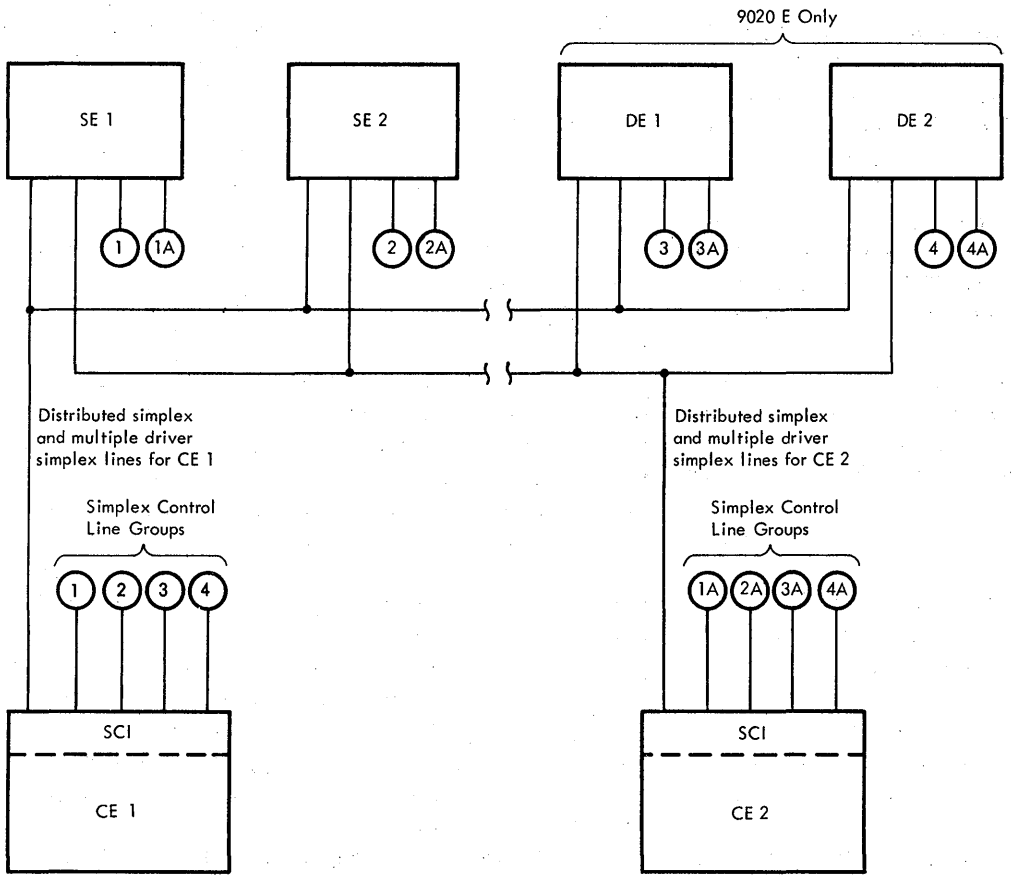


Figure 2-60. Basic Storage Configuration

The term "multiple driver simplex" refers to lines going from all SEs and DEs in the system to one CE. The functions of the bus and control lines are defined below.

Simplex Control Lines

Select Storage Odd. This signal is activated by a CE to request a storage cycle in the odd BSM. (SAB bit 20 = 1 has been decoded by the SCI to indicate an odd doubleword address.)

Select Storage Even. This signal is activated by a CE to request a storage cycle in the even BSM. (SAB bit 20 = 0 has been decoded by the SCI to indicate an even doubleword address.)

Logout Stop. This control line is pulsed by a CE in response to an 'SE' or 'DE check' signal or an 'element check' (ELC) signal and during a diagnose 'SE logout'. The 'logout stop' causes the SE or DE to halt all activity at the end of the cycle in progress and to activate logout priority for the CE controlling the logout.

Logout Select. This is a special signal provided by the CE to request a doubleword of logout data from an SE or DE.

Logout Complete. This control line is activated when a CE completes logout of an SE or DE, or during a subsystem reset. The 'logout complete' resets the check condition in the associated SE or DE.

Reconfigure Select. When executing a SCON instruction, the CE activates 'a reconfigure select' signal to each SE and DE selected by the selection mask. This signal causes the SE or DE to gate the contents of SDBI to its configuration control register.

Suppress Log Check. This control line is activated by a CE along with a 'select odd' or 'select even' signal; it causes suppression of a 'data check' or an 'ELC' signal during that cycle. It is used while logging out the ST register, which may have bad parity.

Request Acknowledged. This control line is not used in the CE.

Accept. This control line is activated by an SE or DE to indicate to a CE that a storage cycle has been started for its request. This signal also sets the CE response latch for logout purposes.

Reconfigure Accept. This control line is activated by an SE or DE to indicate to a CE that the contents of the SDBI have been loaded into the SE or DE CCR and that no parity errors exist.

Logout Advance. This control line is activated by an SE or DE in response to 'logout select' from a CE when logout data has been placed on SDBO.

Element Check (ELC). This control line is activated by an SE or DE to alert all CEs that the SE or DE has an error condition. The 'ELC' signal may be a pulse or a level, depending on the error condition:

1. Pulsed ELC:
   a. CCR parity error.
   b. Temperature marginal.
   c. On battery.
   d. Storage checks (mark parity, address parity, key parity, data parity, double cycle timeout, or box tag mismatch).
2. Level ELC:
   a. Over or under voltage, or overcurrent condition.
   b. Power off.
   c. SE Stopped.

SE Ready. This control line is activated by an SE or DE to indicate to the CE that storage is available for use, i.e., power on, configured to the CE, and not in Test mode.

SE Stopped. This control line is activated by an SE or DE to indicate that it is being logged out. All operations of the SE or DE are inhibited, except logout and reconfiguration.

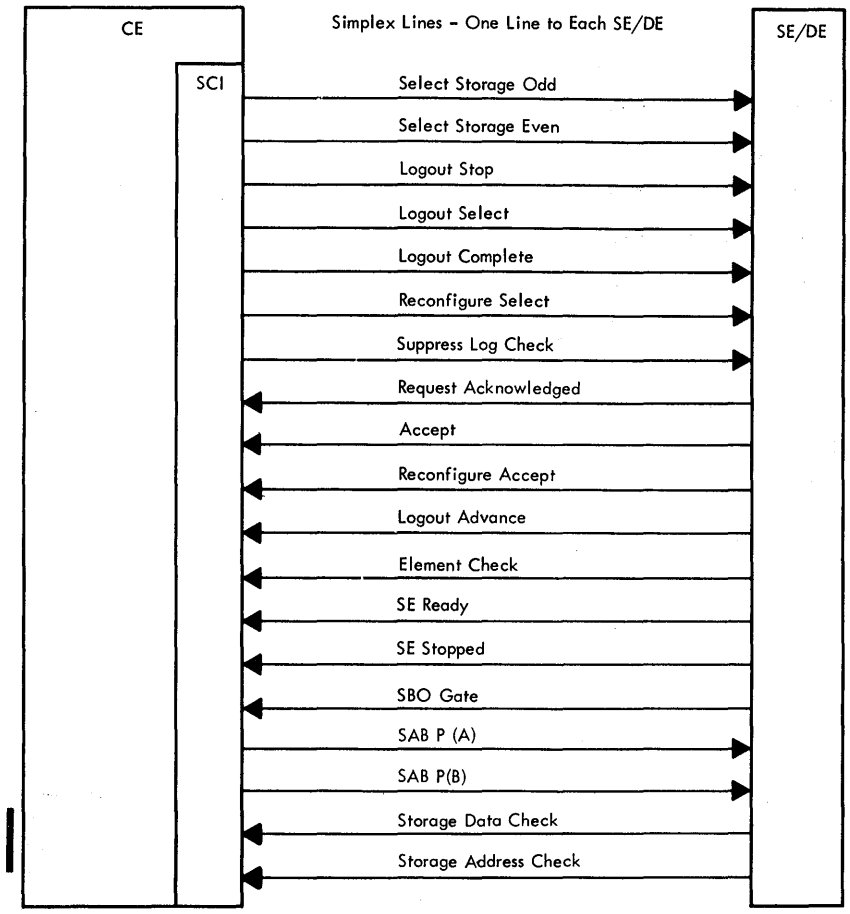
SBO Gate. This control line is activated by an SE or DE to identify the source of data on the SDBO during a fetch cycle or to identify the source of a check signal when an error is detected by the SE or DE.

SAB P_A. This line carries the parity bit generated in the SCI for SAB (6–12).

SAB P_B. This line carries the parity bit generated in the SCI for SAB (13–19).

Storage Data Check. This control line is activated by an SE or DE when a parity error is detected in its data register during a store, test and set, or fetch operation or when a multiaccept error is detected. (A multiaccept condition occurs when two or more accept lines are activated at the same time.)

Storage Address Check. This control line is activated by an SE or DE on a multiaccept condition or when a parity error is detected in one of the following SE or DE registers: (1) the address register during a store, test and set, or fetch operation; (2) the mark register during a store or test and set operation; (3) the storage protect address register during a store, test and set, fetch, set key, or insert key operation; and (4) the key register during a store, test and set, fetch, or set key operation. It is also activated if the 'normal op' control line is active during a test and set, set key, insert key, double cycle, or suppress log check operation or inactive during a fetch or store operation.



●Figure 2-61. Storage Interface Lines (Sheet 1 of 3)

Distributed Simplex Lines

Storage Address Bus (SAB). This bus (19 address lines and 1 parity line) designates the address of a doubleword in main storage. (Two additional SAB parity bits are sent on simplex lines.) Bits 1–4 (referred to as the box tag) designate the SE or DE, and bits 4–19 designate the doubleword within the odd or even portion of the SE or DE. (The odd or even portion of the SE or DE is selected in the CE by SAB bit 20.)

Storage Data Bus In (SDBI). This bus (64 data lines and 8 parity lines) carries data sent from the CE to the SEs and DEs.

Mark Bus. This bus (eight control lines and one parity line) designates which data bytes on the SDBI are to be stored into the selected SE or DE; a mark line corresponds to each byte on the SDBI. No signals are on the mark bus during a fetch operation.

In Key Bus. This bus (five key lines and one parity line) is used during fetch, store, and set-key operations to transfer the storage protection key from the CE to the storage protection area in the selected SE or DE.

Set Key. This control line is activated by a CE causing the selected SE or DE to store the contents of the In Key bus into its storage protection area.

Insert Key. This control line is activated by a CE causing the selected SE or DE to outgate the specified storage protection key to the Out Key bus.

Store. This control line is activated by a CE to permit an SE or DE to store data.

Test and Set. This control line is activated by a CE to cause the selected SE or DE to perform a test-and-set cycle.

Note: If none of the above control lines are active during a 'select odd' or 'select even' pulse time, a fetch data cycle will result.

Cancel. This control line is activated by a CE when it detects an invalid address condition or a specification error. This signal causes the SE or DE to regenerate the data read from core storage and prevents data transfer to and from the CE.

Defeat Interleave. When this control line is activated by a CE, it causes the SCI to interchange SAB bits 20 and 6.

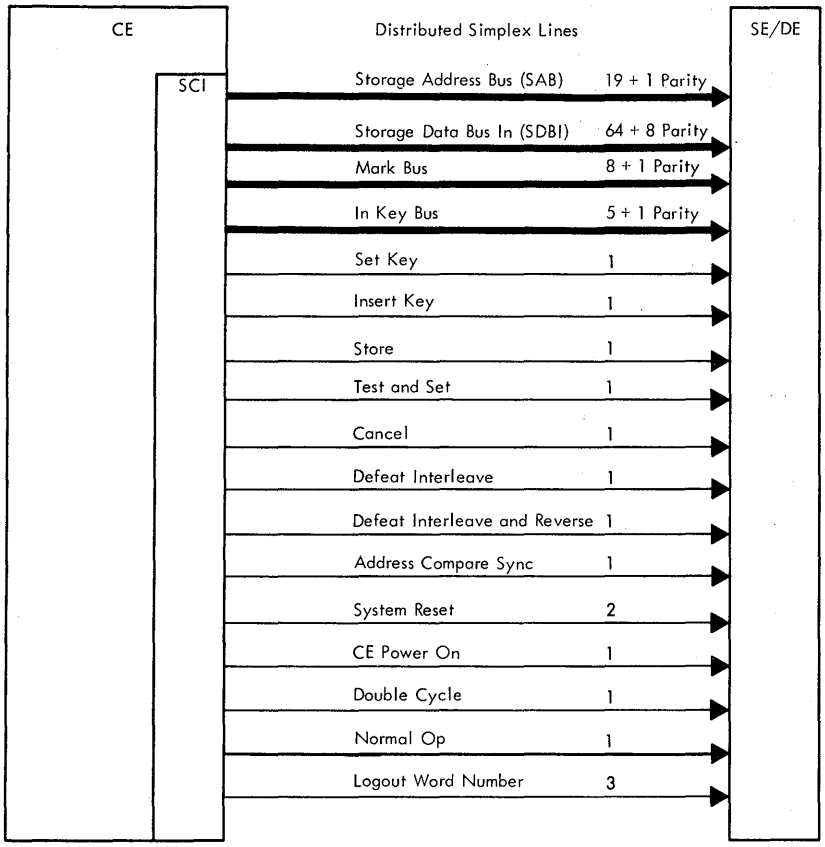


Figure 2-61. Storage Interface Lines (Sheet 2 of 3)

This has the effect of causing requests or stores to be taken from consecutive even addresses if SAB (6) is 0, or consecutive odd addresses if SAB (6) is 1.

Defeat Interleave and Reverse. This control line has the same effect as 'defeat interleave' except the value of SAB (6) is reversed; i.e., odd addresses are requested if SAB (6) is 0, and even addresses are requested if SAB (6) is 1.

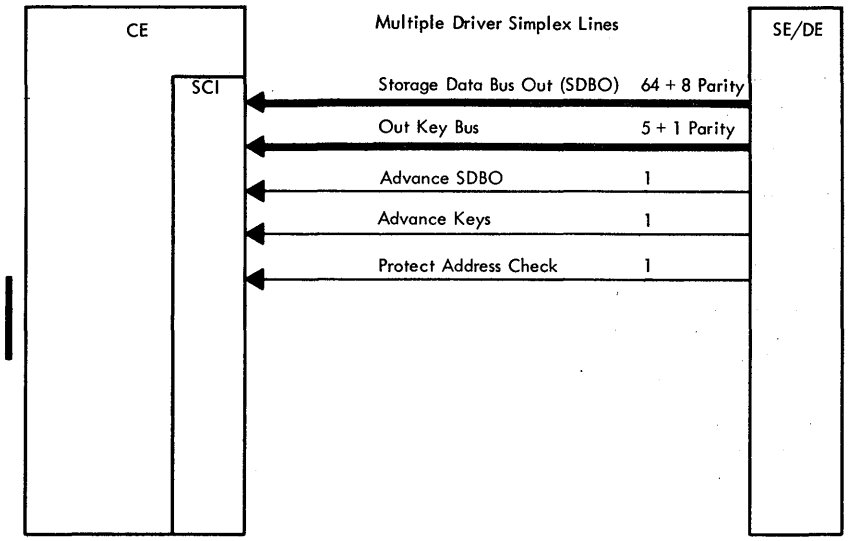
Address Compare Sync. This line provides a signal to all SEs and DEs when the MAIN STORE ADDRESS COMPARE key setting (on the CE control panel) matches the SAB and the ADDRESS COMPARE switch is in the PROC position.

System Reset. Two lines are activated by a system reset condition in any CE. They cause a storage reset in all SEs and DEs regardless of their CCR setting. Line 1 must be minus, and line 2 must be plus to generate a storage reset. The CCR communication bits in each SE and DE are set to 0, and the SCON bits are set to ones (good CCR parity is preserved).

CE Power On. This control line goes to ground level just before CE power goes off; it remains there until after CE power-on reset. It inhibits the output of the CCR communication bit for that CE in all SEs and DEs.

Double Cycle. This control line is activated by a CE when executing the logical AND, OR, or Exclusive OR immediate instructions. It has no effect at the DE, where the line is used only for checking the 'normal op' line. However, at the SE, 'double cycle' ensures that no other CE or IOCE will alter the data between the two accesses required to perform the AND, OR, or Exclusive OR. This is accomplished by preventing the SE from giving priority to another CE or IOCE until two successive storage cycles have been completed. To ensure that a double cycle is obtained, the CE must issue the second select within 5 usec after the first storage cycle is completed. Note that the three logical instructions mentioned may be executed on data in a DE with no difference in SCI operation. The double cycle protection is not implemented in the DE however.

Normal Op. This control line is activated by the CE as an interlock to prevent control-line-driver or receiver failures from causing unwanted storage cycles or multiple exceptions (such as simultaneous fetch and set key operations, which could result in destroyed storage data). It must be active with fetch and store operations and inactive with test and set, set key, insert key, double cycle, and suppress log



●Figure 2-61. Storage Interface Lines (Sheet 3 of 3)

check operations. Violation causes the SE or DE to stop, 'storage address check' to be sent to the using CE, and the 'element check' signal to be sent to all CEs in the system.

Logout Word Number. These three lines are used by the CE to control the gating of logout words to the SDBO during logout of an SE or DE.

Multiple Driver Simplex Lines

Storage Data Bus Out (SDBO). This bus (64 data lines and 8 parity lines carries data sent from the SE or DE to the CE.

Out Key Bus. This bus (five key lines and one parity line) is used only during the insert-key operation that transfers the storage protection key from an SE or DE to a CE.

Advance SDBO. This control line is activated by an SE or DE before data is gated to the SDBO.

Advance Key. This control line is activated by an SE or DE before key data is gated to the out key bus on an insert key cycle.

Protect Address Check. This control line is activated by an SE or DE when the key supplied by the CE does not match the key of the block accessed.

Basic Operating Considerations

- Requests are issued by: (1) I-fetch, (2) microprogram, (3) scan.
- CE clock is stopped at the end of the cycle following a 'select' cycle.
- SCI interleaves 'odd' and 'even' selects.
- Storage keys protect storage contents.
- Insert key operation fetches key to CE.
- Set key operation replaces key in main storage.
- Logical and physical PSBARs specify PSA for a CE.
- Contents of ATR slot are decoded to select SE or DE.

The SCI resides in, and is a logical but independent part of, the CE. To enable efficient handling of main storage requests, the SCI and CE clocks are synchronized. The SCI is designed to accept storage requests from the CE every 400 ns (every other clock cycle). However, the speed with which these requests are serviced depends on the priority logic within the SE or DE, which must handle requests from other CEs and IOCEs in the system. (The term "main storage" refers to all the SEs and DEs in the system.)

The CE can issue five types of requests to the SCI: fetch data, store data, insert key, set key, and test and set. The SCI decodes the address supplied by the CE and issues a select to the proper SE or DE. The SCI stops the CE clock two cycles after issuing select to main storage and restarts it upon receiving an 'accept' response.

Figure 2-62 shows the basic organization of storage elements and display elements. Interface lines from CEs and IOCEs connect to the storage switching unit (SSU) in the SE and to the switch unit (SU) in the DE, which contain logic that maintains a priority scheme. If a CE or IOCE issues a request to a busy SE or DE, priority is granted after all higher-priority requests have been serviced. The request is sent through common logic to the basic storage module (BSM) selected. The BSM operates independently after it receives a request. This allows interleaving between an odd and an even BSM. The request is also sent through common

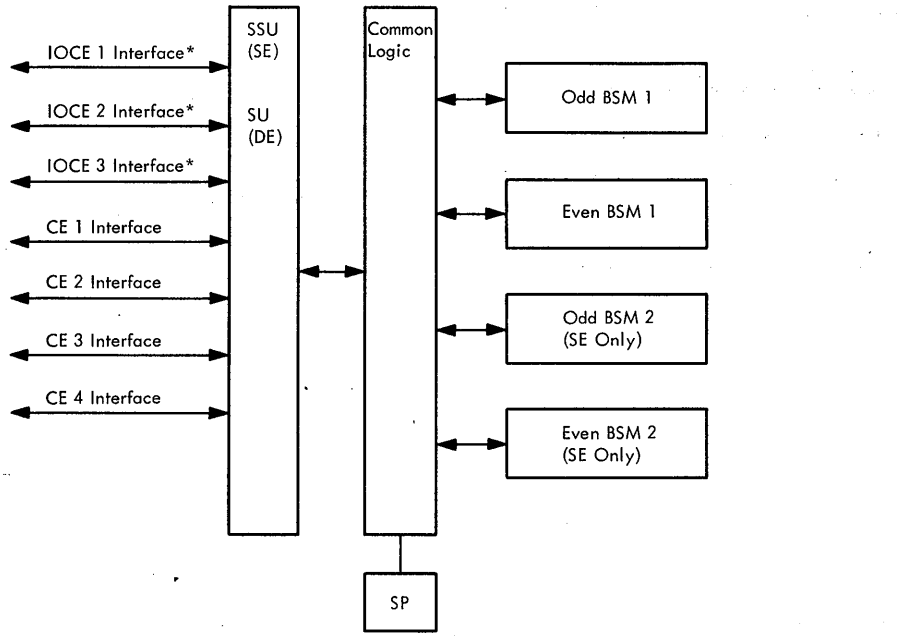


Figure 2-62. Basic Organization of a Main Storage Element (SE or DE)

logic to the storage protect (SP) unit to start an SP cycle. Protect key comparison is made in the common logic.

A need to fetch new data is detected in the CE three or four cycles before this data is required. Accordingly, a three- or four-cycle fetch request is issued to the SCI, indicating that data will be required after three or four cycles have elapsed from the time of the request.

Requests to store data are initiated in the CE by the microprogram or by the scan controls; these requests are always accompanied by the 'mark' signals, which designate the bytes to be stored. Store requests are not categorized into the three- and four-cycle types since no critical transfer into the CE is involved; once the SE or DE is selected, the CE no longer depends upon the SE or DE operations.

The insert key, set key, and test and set requests are issued by the CE microprogram. Basically, the insert key request is a fetch operation to obtain the protection key from main storage. The set key request is a store operation which transfers the five-bit (plus parity) storage protection key from the CE into a specified storage protect area of main storage. The test and set request is essentially a combined fetch/store operation effected in a single storage cycle.

A storage protection capability is provided to prevent the contents of main storage from being used or destroyed by mistake. Both fetch and store operations are subject to this protection. Protection is accomplished by dividing each SE and DE into 2048-byte blocks and assigning a one-byte protection key pattern to each block. The assigned key patterns for all 2048-byte blocks within a storage element are recorded in the storage protection (SP) unit of that element. During a storage request, the SP unit compares the key pattern for the addressed block with the key pattern supplied by the requesting CE. (Keys are gated from the SCI over the 'in key' bus during the storage request.) A mismatch in keys results in a protection violation; the storage request is not honored, and a 'protect address check' signal is sent to the requesting CE. (A zero key from the CE or from the SP unit will result in a match.)

The protection key for any 2048-byte block of storage can be fetched from the SP unit and brought into the CE for inspection. This operation is performed through execution of the Insert Storage Key (ISK) instruction, which issues a fetch request directly into the SP unit of the storage element. Conversely, the protection key for any 2048-byte block of storage can be changed through execution of the Set Storage Key (SSK) instruction by a CE. This instruction provides a new key pattern and issues a store request into the SP unit.

The ISK and SSK instructions can be executed only in the PSW Supervisor state and are not available to the problem programmer. A programmed protection of storage can be achieved by means of the Test and Set (TS) instruction. When the TS instruction is executed by a CE, a doubleword is fetched from main storage, and the CE inspects the addressed byte of that doubleword. The SE or DE sets the addressed byte to all-1's, while the remaining bytes in the doubleword are not changed. Thus, programmed storage protection is achieved (in the sense that a CE can later inspect the first byte of a particular storage block to establish whether this block has been previously processed).

A test and set storage request combines aspects of both fetch and store operations; therefore, it requires special handling by the SCI and the SE and DE. From the SCI viewpoint, the TS instruction is the only one that generates a 'mark' signal during a fetch request. The SCI sends the 'mark' signal and the 'test and set' signal to the SE to DE. During the storage write cycle, all bytes except the byte specified by the 'mark' signal are regenerated into the core array; all-1's are generated into the byte location specified by the 'mark' signal.

The preferential storage area (PSA) of a CE is a 4096-byte block of main storage that a CE must have to maintain its operation. Because of the flexibility required of the 9020 system, it is necessary to be able to designate any 4096-byte block of main storage in any SE (a DE may not be used) as a PSA for a CE and to be able to make a reassignment as the need arises. Two registers, logical preferential storage base address register (logical PSBAR) and physical PSBAR, contain the address of the PSA; reassignment is accomplished by executing a Load Preferential Storage Base Address (LPSB) instruction. The assignment must be made to an SE that is configured to the CE (communication bit on for that SE).

The address translation register (ATR) is a 40-position register in the CE that is logically divided into ten four-position "slots". The ATR provides the capability for the CE to relocate a span of logical main storage addresses, on SE and DE boundaries, to any SE or DE configured to it by changing the contents of the ATR.

The term "logical address" refers to an address as designated by a program; "physical address" refers to an address that has been translated by the ATR to designate a particular SE or DE. The four high-order bits of the logical address select an ATR slot, and the contents of the ATR slot select the physical address of an SE or DE.

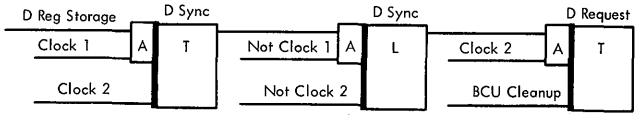
Basic Control and Timing Considerations

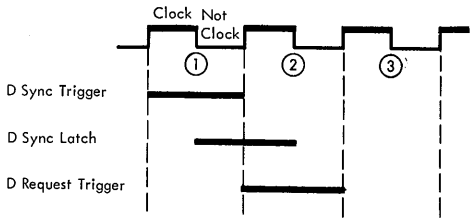
- Requests are recorded by sync trigger/latches.
- SCI decodes address supplied by CE to issue an odd or even select to SE or DE.
- 'Select' signal initiates a storage cycle in SE or DE.
- Access time to a nonbusy SE or DE is approximately 600 ns.

The scheme for processing storage requests by the SCI is shown in Figure 2-63. Requests are entered into the SCI request sensing logic and are recorded by the corresponding sync trigger/latch circuits of the SCI. The storage address is then gated from the IC, D, or PAL to the SCI for decoding. At completion of the decoding, the SCI initiates a storage cycle by gating the storage address to the storage address bus (SAB) and sending a 'select odd' or 'select even' signal to the SE or DE. Priority logic in the SE or DE allows an 'accept' response when no higher-priority CEs, IOCEs, or CVGs (DE only) are requesting service. When the SCI receives an 'accept' response, it restarts the CE clock and continues processing data.

On a fetch data request, the SE or DE gates one doubleword of data into the SDBO for the requesting CE. On a store data request, the SE or DE replaces the contents of the addressed location with the data sent over the SDBI from the CE. Only those bytes designated by the 'mark' signals are placed into main storage; in the absence of 'marks', bytes already in storage remain unaltered.

To provide the necessary control signals at the correct time, the SCI makes extensive use of trigger/latch circuits. SCI triggers are set at clock time of the machine timing signal and are reset at the following clock time. Conversely, the latch circuits are set at not-clock time of the timing signal and are reset at the following not-clock time. Thus, SCI timing sequences are carried out by sequential shifting of status information through latch-to-trigger-to-latch circuits. A typical arrangement is shown below. Note that the state of a particular trigger or latch at any particular time is indicative of the progress made since the issue of the request:





As mentioned previously, the CE can generate either a three- or four-cycle fetch request to specify the exact time at which the SDBO is to be gated to the CE. Data will be available on SDBO after three cycles have elapsed from the time of 'accept'. To meet the requirement of a four-cycle request, the SDBO is held valid for an additional machine

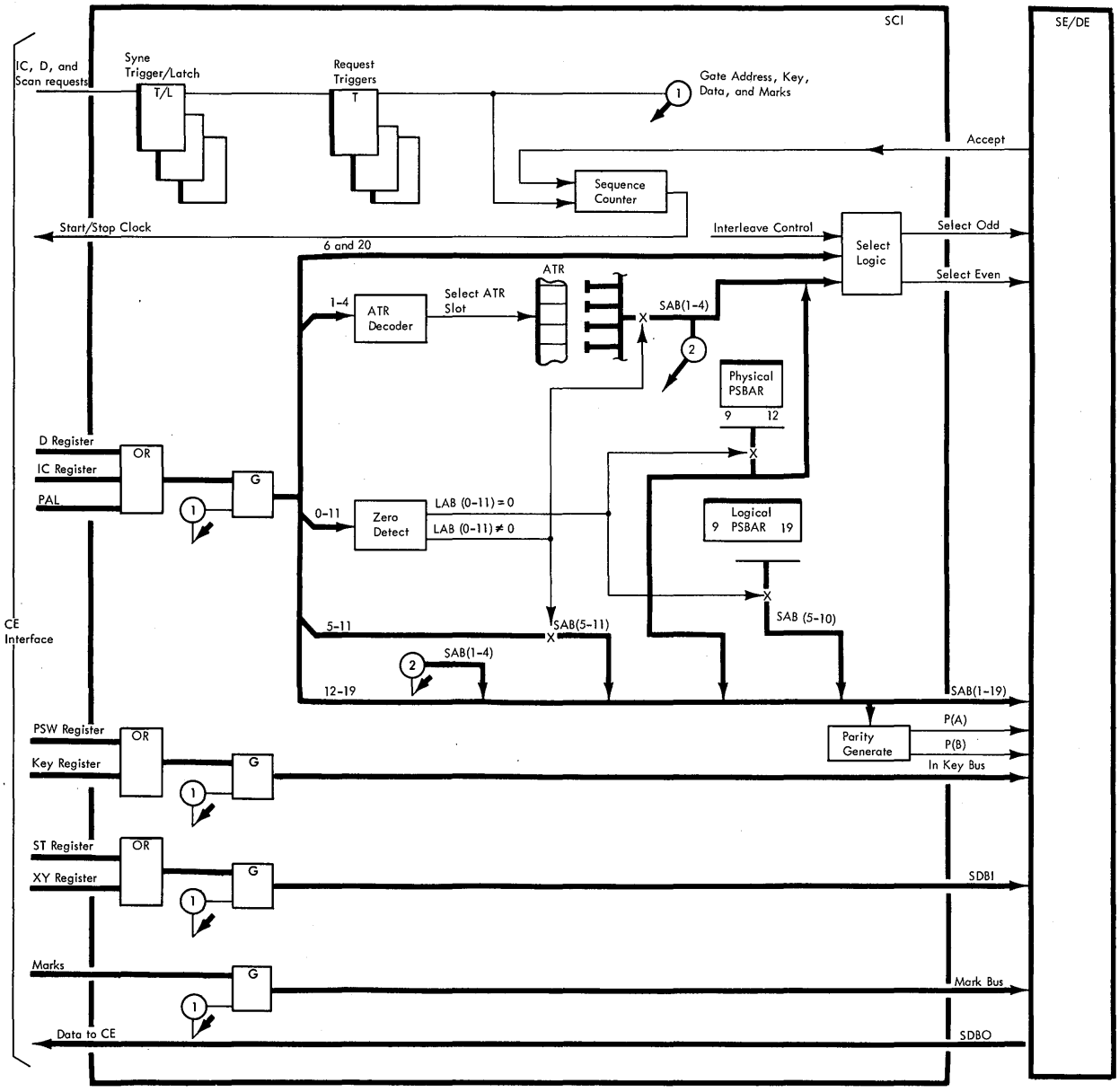
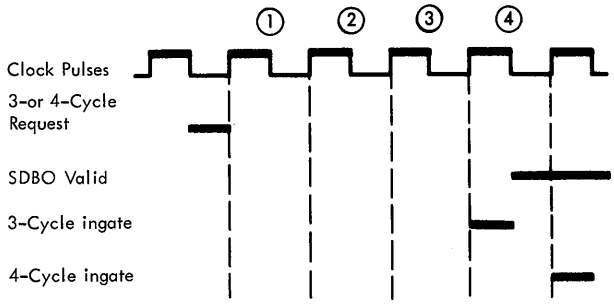


Figure 2-63. Basic Scheme for Processing Storage Requests

cycle. This provides for data to be present at the SDBO during the four-cycle sample pulse:



Basic Operational Sequence

- CE sequencers count CE cycles following a storage request.
- Address is developed and sent to storage.
- 'Select odd' or 'select even' is generated if selected SE or DE is ready and not stopped.

The basic operational sequence of handling storage requests is shown in Figure 2-64.

Storage requests are issued by one of three functional areas within the CE: I-fetch logic, ROS microprogram, or scan controls. Once the request is issued, a group of CE sequencers is activated in the SCI. These sequencers provide for stopping the CE clock two cycles after 'select' is issued to storage.

During the following discussion, be aware of the two numbering schemes that exist for main storage address bits. D and IC registers (sources of main storage addresses) are displayed on the CE console as bits 8–31, but they are referenced in the ALDs as bits 0–23. In this discussion, all reference to storage address bits and related SCI logic is consistant with ALD reference (using the 0–23 scheme). Chart 2 of Diagram 4-602, FEMDM shows how the two numbering schemes relate to each other.

The outputs of the IC, D, or scan sync latches and request triggers are used to gate an address from D or IC (0–11) or from PAL (40–51) to logical address bus (LAB) bits 0–11. If LAB (0–11) is equal to 0, physical PSBAR (9–12) and logical PSBAR (13–19) are gated to SAB (1–11). If LAB (0–11) is not equal to 0, LAB (1–4) is used to select an ATR slot, which contains the physical address of the element to be selected. The physical address is gated to SAB (1–4), and D or IC (5–11) or PAL (45–51) are gated to SAB (5–11). SAB (12–23) is always gated from D or IC (12–23) or PAL (52–63). SAB bits 1–9 are sent to the SE or DE for decoding and checking.

The 'select' signal is developed by decoding SAB (1–4) to determine the element, and SAB (2) determines whether the select is even or odd. If the selected SE or DE is stopped or not ready (power down or not configured), an 'invalid address' signal will be developed.

DETAILED ANALYSIS OF SCI FUNCTIONS

For purposes of discussion, the SCI is divided into a number of functionally distinct logic areas. The subsequent paragraphs describe the functions performed by each area and explain how these functions fit into the overall operational sequence of the SCI.

The following discussions reference functional diagrams in the companion FEMDM. These diagrams are based on the ALDs and maintain the same line terminology as the ALDs. In the ALDs, the SCI is referred to as the Bus Control Unit (BCU), and the CE is referred to as the Central Processing Unit (CPU). To conform with the ALDs, the terms BCU and CPU also appear on some diagrams of the FEMDM.

Initial Handling of Requests

- CE requests during clock time are recorded by sync triggers.
- CE requests during not-clock time are recorded by sync latches.

The SCI logic used for initial sensing and for recording main storage requests is shown in Diagram 4-601, FEMDM.

Storage requests from the CE can be issued to the SCI at either clock or not-clock time of the machine cycle. To synchronize the clock and not-clock requests, the SCI employs a trigger/latch sync arrangement. Requests received at clock time are first entered into the SCI sync triggers; they are then propagated (at not-clock time) into the sync latches. Requests received at not-clock time are entered directly into the sync latches. Thus, at the completion of one machine cycle, all requests are reduced to a common time-reference frame.

From the SCI "viewpoint", the storage requests issued by the CE can be placed into one of three general categories:

1. Requests Generated by Microprogram. These requests are decoded at clock time and, depending on the address source (IC or D), are entered into the corresponding SCI sync triggers. Furthermore, all fetch requests must be specified as being either 3 or 4 cycles in duration. This is to inform the SCI of the specific time at which the requested data must be gated into the CE. The presence

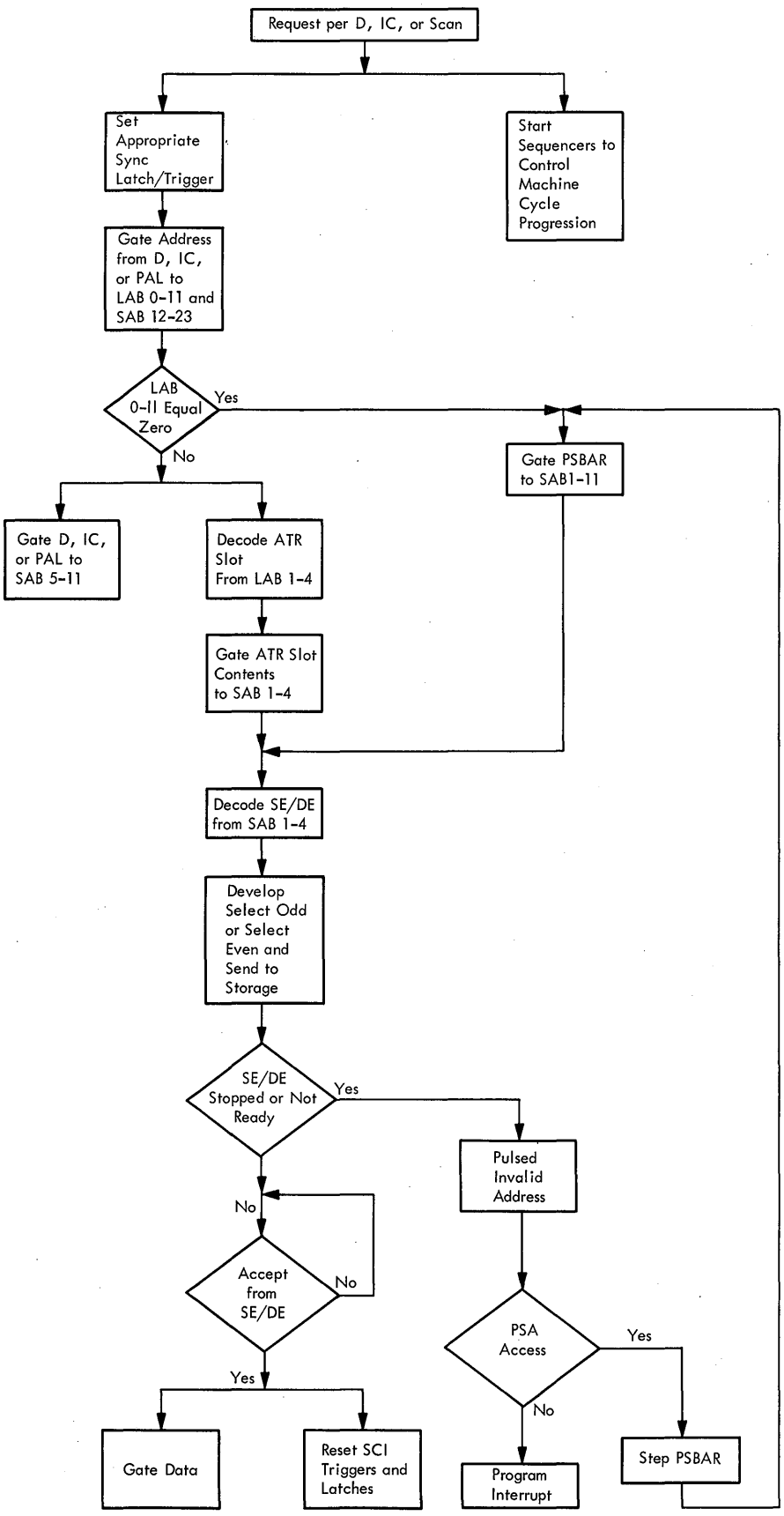


Figure 2-64. Basic SCI Operation

or absence of the '3-cycle access' signal from the CE indicates whether a three- or four-cycle fetch has been initiated; i.e., the '3-cycle sync' trigger (in the SCI) is set on all 3-cycle fetch requests and reset on all four-cycle fetch requests. Upon the setting of the appropriate sync trigger in the SCI, the request is propagated (at not-clock time) into the corresponding sync latch.

2. Requests generated by I-fetch hardware. These requests are decoded at not-clock time and, therefore, are entered directly into the corresponding IC or D latch and the '3-cycle sync' latch.
3. Requests generated by scan controls. These requests, generated during logout and ROS test operations, are decoded at clock time of the machine cycle. Accordingly, the requests are first entered into the scan-sync trigger and are then propagated into the 'scan-sync' latch at not-clock time. Fetch requests initiated by scan operations are always specified as four-cycle requests; i.e., the '3-cycle' trigger is not set.

At clock time of the machine cycle following the requests, the signals from the sync latches are further propagated into the appropriate request triggers and into the 'CPU request' trigger. They also feed the CPU sequencer and clock control logic, which stops the CE clock two cycles after select is issued.

D, IC, or PAL is gated to SAB according to the type of request, i.e., from D for a request per D, from IC for a request per IC, and from PAL for a request per D or IC when PAL is gated to D or IC. The latter case allows PAL to be gated to SAB and to D or IC at the same time.

In conjunction with a request per D, the CE may issue an 'insert key', 'set key', or 'test and set' signal to the SCI. These signals are recorded into the appropriate SCI triggers and are later used to modify the handling of the storage request. Basically, these modifications are as follows:

1. The 'insert key' trigger causes an 'insert key' signal to be sent to storage during the handling of the request.
2. The 'set key' trigger issues a 'set key' signal to storage and gates the F register (0–3) to the 'in key' bus.
3. The 'test and set' trigger causes a 'test and set' signal to be sent to storage during the handling of the request.

If none of the above three triggers are active, the 'normal operation' control line to all SEs and DEs in the system is activated.

If the CE decodes an operation code of 94 (AND), 96 (OR), or 97 (XOR), the SCI sets the 'double cycle' trigger. This makes the 'double cycle' interface line active, which conditions priority logic in the SE or DE to allow an additional storage cycle without interference from another CE or IOCE.

The 'XY sync' latch is set by a micro-order and controls gating to SDBI: if the latch is set, data is gated from the XY register; if the latch is not set, data is gated from the ST register.

During a machine check logout, the 'suppress log check' signal is activated as log word 23 (ST register) is stored. This prevents a storage data check if the ST register has a bad parity condition.

Address Decode and Gating

- Storage address is gated from D, IC, or PAL.
- If 'gate PSBAR to SAB' is active, logical PSBAR and physical PSBAR are gated to SAB (1–11).
- If 'gate PSBAR to SAB' is not active, LAB (1–4) is decoded to gate an ATR slot to SAB (1–4).
- SAB (6) and SAB (20) are used in conjunction with DEFEAT INTERLEAVING switch to develop 'decode old address' or 'decode even address' signals.

Diagram 4-602 shows the various address translation functions performed by the SCI. The diagram includes two charts: chart 1 shows the range of addresses for SEs and DEs, and chart 2 shows the CE console labeling for D and IC register display translated to CE logic line labeling; e.g., CE console D register bit 8 corresponds to CE logic D register bit 0.

D, IC, or PAL is gated to logical address bus (LAB) bits 0–11 and to the internal storage address bus (SAB) bits 5–23. (This discussion refers to the internal/external SAB to distinguish between the SAB bits in the SCI logic and the SAB interface lines to storage.)

If the request is made to a preferential storage area, LAB (0–11) equals 0, and 'gate PSBAR to SAB' is activated. This control line gates physical PSBAR (1–4) to SAB (1–4) and logical PSBAR (13–19) to SAB (5–11). If the request is not made to a preferential storage area, 'gate PSBAR to SAB' is not activated'. In this case, the ATR slot is decoded from LAB (1–4), and its contents are gated to SAB (1–4); logical PSBAR (13–19) is gated to SAB (5–11), and SAB bits 12–19 are sent directly to SAB (12–19).

Interleaving of storage requests is accomplished by alternating requests between odd and even basic storage modules (BSMs) within the same element. This allows a storage cycle to be started before the preceeding cycle has been completed. SAB (6) and (20) are decoded in the SCI to generate selects to the odd/even BSMs. The manner in which the SCI decodes these bits depends on the position of DEFEAT INTERLEAVING switch (see Figure 2-65). When the switch is in the normal (PROC) position, the

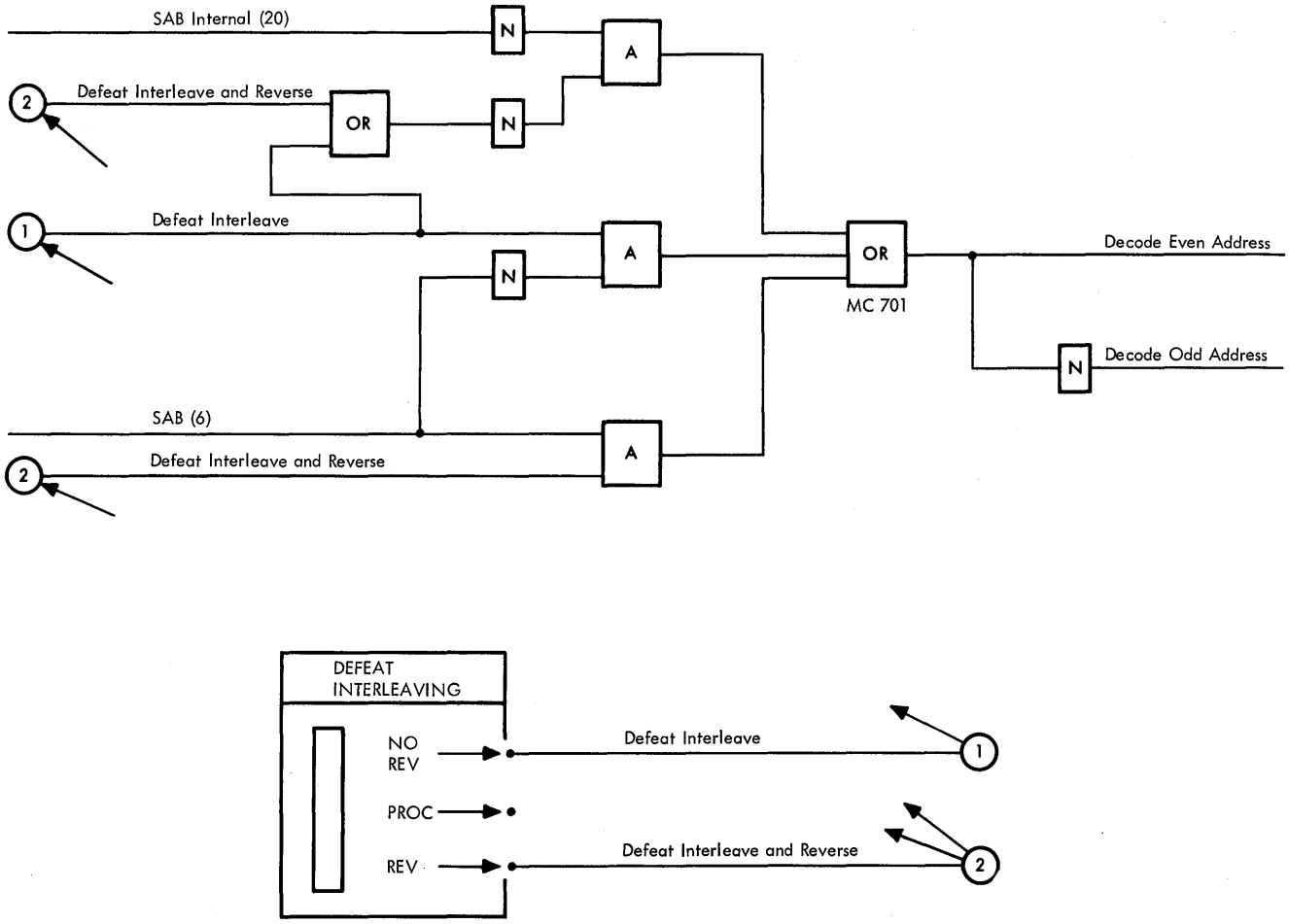


Figure 2-65. Decode Odd/Even Address

'defeat interleave' line is not active, and SAB (20) determines even or odd storage BSM (even if SAB internal (20) equals 0, odd if SAB internal (20) equals 1). When the switch is set to the REV position, the function of SAB (6) is reversed (selects even if SAB (6) equals 1, selects odd if SAB (6) equals 0). When the switch is set to NO REV, SAB (6) determines the even or odd storage area (selects even if SAB (6) equals 0, selects odd if SAB (6) equals 1). In both the REV and NO REV positions, the internal SAB (20) is sent to storage as SAB (6).

Select to Storage

- 'Decode odd address', 'decode even address', and SAB (1–4) activate a 'select' line to storage.

The storage frame is decoded from SAB (1–4) as shown in Diagram 4-602, FEMDM. If the decoded SE or DE is configured, ready, and not stopped, the 'select timing pulse', 'SE decoded', and 'decode even address' or 'decode odd address' are ANDed to activate one 'select' line to storage.

The 'storage 2' trigger is turned on by D, IC, or 'scan request' trigger and is reset on the following clock cycle. The 'select timing pulse' is activated by the 'BCU oscillator' when 'storage 2' trigger is on; it provides a means of timing the 'select' pulse with the address bus at the SE or DE. Timing adjustment is achieved by means of delay tap selection in the select timing pulse circuitry.

Stopping the CE Clock

- CE sequencers control distribution of clock timing within CE.
- CE sequencers are started at clock time of cycle following request.

- Sequence in which sequencers are stepped varies with request being processed:
  1. Four-cycle fetch: 'CPU 2' trigger, 'CPU 2' latch. 'CPU 3' trigger, 'CPU 3' latch, 'CPU 4' trigger, 'CPU 4' latch, 'CPU 5' trigger, 'CPU 5' latch.
  2. Three-cycle fetch: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger, 'CPU 4' latch, 'CPU 5' trigger, 'CPU 5' latch.
  3. Store or set key: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger.
  4. Insert key: 'CPU 5' trigger, 'CPU 5' latch.

- Conditions that stop CE clock:
  1. 'CPU 2' latch is set.
  2. Insert-key operation.

A group of CE sequencers (four trigger/latch combinations) is used in the SCI to control CE cycle progression after each storage request. These sequencers control the 'stop CPU clock' trigger in the SCI. The 'stop CPU clock' trigger has direct control of the CE clock: setting this trigger stops the CE (on the following cycle); resetting this trigger starts the CE (on the following cycle). Diagram 4-603, FEMDM, shows the CE sequencers and the control logic for the 'stop CPU clock' trigger. The sequencers are started on the CE cycle following a storage request and are advanced by the subsequent CE clock signals. The 'CPU 2' trigger and latch are set during the first CE cycle following the request; the 'CPU 3' trigger and latch are set during the second CE cycle following the request, and so on.

The 'stop CPU clock' trigger is designed so that if its reset logic is active its set logic is prevented from setting the trigger. Because of this method of implementation (Diagram 4-603), if the 'CPU 2' latch is set (first cycle following a storage request), the 'stop CPU clock' trigger will be set on the next cycle (second cycle following the request), and the CE clock will be inhibited from performing the third processing cycle. This clock stopping sequence occurs during both fetch and store operations, retaining the storage address in the IC, D, or scan controls until the SE or DE responds with an 'accept' signal. The 'stop CPU clock' trigger is reset by the 'BCU cleanup select successful' signal, the CE clock is started, and both CE processing and further sequencer stepping are continued.

For store data operations, the CE sequencers are started in the normal manner. However, the output of the 'store' latch modifies the subsequent sequencer stepping as follows: 'CPU 2' trigger and 'CPU 2' latch are set on the first cycle following the request; 'CPU 3' trigger is set on the second cycle following the request. Further sequencer advance is inhibited during store operations because ingating is not required.

Detection and Handling of Invalid Address

- SE or DE not configured.

- SE or DE has power down or is in state zero with TEST switch set to ON.

- SE or DE stopped for logout.

- Force 'select' to PSA SE for resetting SCI logic.

An invalid address condition is detected in the SCI logic if an attempt is made to select an SE or DE that is stopped or not ready (power down, not configured, or in state zero with TEST switch on). The invalid address detection logic, with an error-handling flowchart, is shown in Diagram 4-604, FEMDM. For detailed timing refer to ALD A7511.

The 'test for invalid address' trigger is turned on for each storage request. The output of this trigger monitors the operational condition of the decoded SE or DE. If the SE or DE is not ready, the 'invalid address' trigger is set; if stopped, the 'SE/DE stopped' trigger is set. Either case results in a program interruption.

When a request trigger is set, 'issue a select' is activated, and the 'storage 2' trigger is set. The output of the 'storage 2' trigger activates 'select timing pulse' and sets 'inhibit storage select' latch to prevent additional 'select timing' pulses. If the selected SE or DE is stopped or not ready, an additional select timing pulse is forced (to reset SCI) when the 'inhibit storage select' latch is turned off. (The request trigger remains on until an 'accept' pulse is received.)

The following discussion considers three cases: (1) a request to a stopped SE or DE, (2) a request to a not-ready SE or DE, and (3) a PSA request to a stopped or not-ready SE (PSA cannot be assigned to a DE).

1. Request to a stopped SE or DE: The 'decoded SE/DE stopped' signal is activated, and it sets the 'SE/DE stopped' latch and trigger. The output of the 'SE/DE stopped' trigger causes a program interruption, sends a cancel signal to all SEs and DEs, and activates a 'select SE even' signal to the SE containing the PSA. The PSA SE responds with an 'accept' signal, but no data transfer takes place because 'cancel' is active. Processing continues.
2. Request to a not-ready SE or DE: The 'invalid address' trigger is set; its output causes a program interruption, causes a 'cancel' signal to be sent to all SEs and DEs, and causes a 'select SE even' signal to be sent to the SE containing the PSA. The PSA SE responds with an 'accept' signal, but no data transfer takes place because 'cancel' is active. Processing continues.

3. PSA request to a stopped or not-ready SE: Program interruption and 'cancel' signal are developed as in the previous two cases. If the 'alternate' latch is on, the CE checks stop and issues a static element check (ELC). If 'alternate' latch is off, it is turned on. The 'gate PSBAR to SAB' signal is active as a result of the PSA request, and it sets the 'PSA request' trigger. The output of this trigger activates the 'step PSBAR to alternate' signal. If inhibit log out stop (ILOS) function is active, the 'step PSBAR' signal is blocked, and the 'select SE even' signal is forced to the same PSA SE. If the SE is still stopped, the CE senses the 'alternate' latch on, checks stop, and issues a static ELC. If the ILOS function is not active, PSBAR steps to the next configured SE (alternate PSA), and a 'select SE even' signal is forced to that SE. If the alternate PSA SE responds with an 'accept' signal, processing continues; if the SE is stopped or not ready, the CE senses the alternate latch on, checks stop, and issues a static ELC.

Storage Timeout

- SE or DE fails to respond to 'select' pulse within allowed time.

A storage timeout condition is detected in SCI logic as shown in Diagram 4-605, FEMDM. The 'select outstanding' signal is activated when a storage request is made to an SE or DE that is ready and not stopped. This signal is ANDed with a '60-cycle' pulse (300-ns pulse every 16 ms) to turn 'latch 1' on. 'Latch 2' is turned on by the output of 'latch 1' when the '60-cycle' pulse becomes inactive. When the SE or DE responds with an 'accept', the 'select outstanding' signal is deactivated, and processing continues. If the 'select outstanding' signal is still active when the following '60-cycle' pulse is activated, a 'storage timeout' pulse is generated to set the 'storage timeout' latch. The time allowed for a timeout check varies from 16 ms to 32 ms and depends on the length of time between 'select outstanding' becoming active and the following '60 cycle' pulse.

The 'set storage timeout on logout' signal (Diagram 4-605) is used to activate the 'storage timeout' pulse, both of the signals are generated only during logout.

SCI Error Handling

- SE and DE can issue:
  1. Storage address check.
  2. Storage data check.
  3. SDBO parity error.
- Errors detected in SCI logic are:
  1. Storage timeout.
  2. Fetch data parity.
  3. SDBI parity.

This paragraph describes the handling of errors detected in SCI logic and error signals received by the SCI from an SE or DE. Reger to Diagram 4-606, FEMDM.

The errors may be divided into two classifications: (1) errors that cause a 'hard stop' condition, and (2) errors that cause a 'check stop' condition. A 'hard stop' condition inhibits the CE oscillator and requires intervention from another CE or from an operator. A 'check stop' condition starts a logout of the CE and causes a machine check interruption.

The 'hard stop condition' latch is set by one of the following conditions:

1. An address is detected outside the PSA during logout.
2. No alternate PSA (APSA) is available.
3. A search for APSA is made when already at alternate.
4. A parity error is detected in logical or physical PSBAR or in the PSBAR counter.
5. A 'PSA lockout' signal is received from an IOCE when the CE is in 360 mode.
6. An 'SE stopped' signal is received from a configured SE when the CE is in 360 mode.
7. An ROS parity error is detected during logout.
8. A 'storage address check' or 'storage data check' signal is received from PSA SE during logout.

The output from 'hard stop condition' latch inhibits the CE oscillator and raises a level ELC to all other CEs. The 'hard stop condition' latch is reset by a 'hard stop reset' or a 'search complete' signal. A 'hard stop reset' signal is activated by a 'logout start' signal from another CE; a 'search complete' signal is activated by a subsystem IPL, a subsystem PSW restart, or an external start from another CE.

'Inhibit clock CE check' (check stop) is activated by an error condition in check register 1 or check register 2. Check register 1 monitors CE parity errors. Check register 2 monitors the following errors:

1. 'Storage error' trigger active.
2. SDBI parity check.
3. LS bus parity check.
4. 'IOCE check response' trigger active.
5. SAB parity check.
6. CCR parity check.
7. ATR parity check.

'Any storage error' and 'storage error' triggers are set by 'storage data check', 'storage address check', 'storage timeout', or 'fetch data check' signals. The output of 'any

storage error' trigger activates the 'storage check step PSBAR' signal and is ANDed with 'not storage timeout' and 'not fetch data check' to send a 'logout stop' (LOS) pulse to the SE or DE that detected the error. (The SE or DE is identified by the 'SBO gate'.)

The 'split log' latch is set if an invalid address, storage timeout, or ROS error is detected during a logout, and it causes a logout to be started in the APSA. The APSA receives a complete set of log data; however, due to a PSA change during logout, this data may not be identical with that in the primary PSA.

PSBAR Operations

- Physical relocation of PSA.
- Search ATR.
- Step to alternate PSBAR.

SCI has the capability of dynamically relocating its preferential storage area (PSA) by "stepping PSBAR". The PSBAR step control logic is shown in Diagram 4-607, FEMDM.

When 'search ATR' signal or 'step to alternate' signal is activated, it ANDs with 'not clock' and 'not end latch or ELC' to start a stepping sequence of 'latch 1', 'trigger 1', 'latch 2', and 'trigger 2'. This sequence is repeated until it is blocked by 'end latch' or 'ELC' becoming activated.

When 'gate new count' signal is not active, it gates an encoded value, which is one greater than the contents of logical PSBAR (9–12), to PSBAR counter (9–12). The one exception is a logical PSBAR (9–12) binary value of 1001 and PSBAR counter (9–12) binary value of 0000.

The 'gate new count' signal is activated when 'trigger 1' is set. This signal gates PSBAR counter (9–12) to logical PSBAR (9–12). The value in logical PSBAR(9–12) is decoded to gate the contents of an ATR slot to physical PSBAR (9–12). The stepping process is repeated until physical PSBAR (9–12) and CCR (8–17) are decoded to activate the 'any frame valid' signal, which sets the 'end latch' to complete the operation. Physical PSBAR (9–12) and CCR (8–17) are also decoded to activate one of ten 'frame valid' signals used during search for a particular SE on a subsystem IPL or a subsystem PSW restart. These signals are ANDed with the 'decode SE' signals, which are decoded from bits 4–7 of MAIN STORAGE SELECT switch. The ANDing of a particular 'frame valid' signal with the corresponding 'decode SE' signal activates the 'SE compare' signal, which sets the 'end latch'.

The 'search ATR' and 'step to alternate PSBAR' operations are shown in Diagram 4-608, FEMDM.

When a subsystem IPL or a subsystem PSW restart is started, logical PSBAR (9–19) is reset, and 'search ATR' signal is activated. The PSA SE is located by comparing MAIN STORAGE SELECT switch setting (4–7) with physical PSBAR (9–12). PSBAR is stepped, beginning with ATR slot 1, until an equal comparison is detected. If no SE is configured to compare with the switch setting, the PSBAR counter is stepped to the binary value 1001, and the 'hard stop condition' latch is set. The CE oscillator is inhibited, and a level ELC is activated to all CEs.

An 'external start' signal resets logical PSBAR (9–19) and activates 'search ATR' signal. A search is made, beginning at ATR slot 1, for the first configured SE. If there is no configured SE, PSBAR counter is stepped to the binary value 1001, and the 'hard stop condition' latch is set. The CE oscillator is inhibited, and a level ELC is activated to all CEs.

'Step to alternate PSBAR' is activated when a storage error is detected in the SE containing the PSA. If the 'alternate' latch is set, or if ILOS is active, the 'hard stop condition' latch is turned on. The CE oscillator is inhibited, and a level ELC is activated to all CEs. If the 'alternate' latch is not set, PSBAR steps, beginning with the current value, until a configured SE is located. If no other SE is configured, stepping is terminated by detecting logical PSBAR (9–12) binary value of 1001 the second time. This sets 'hard stop condition' latch, inhibits the CE oscillator, and activates a level ELC to all CEs.

Page Control

- A page of data is on 512-byte bounds.
- Page controls provide linkage to page overflow address.

A page of data is a maximum of 512 bytes stored on 512-byte bounds. Page controls provide a means of linking two pages when fetching data for the 9020E Display instructions. (See Section 10, Chapter 3.)

The address of the next page is specified in bits 40–60 of the last doubleword in a page. If a '512 carry' is detected in PAL as the current data address is incremented by 8, the page control logic blocks the normal resetting of SCI logic, forces an additional fetch cycle to the next sequential doubleword, and gates SDBO(40–60) to D or IC (Figure 2-66).

The page control hardware is shown in Diagram 4-609, FEMDM. A '512 carry sample page' signal is activated by a '512 carry' in PAL and is ANDed with 'page on next D request' and 'page on next IC request' signals to set 'page/D' latch or 'page/IC' latch. 'Page 1' latch is set to activate 'inhibit BCU cleanup successful', which has the effect of forcing an additional fetch data cycle to the next sequential doubleword. 'Page 2' latch is set when 'early accept' signal is received from the DE and provides a gate for SDBO(40–60) to D or IC. A doubleword boundary

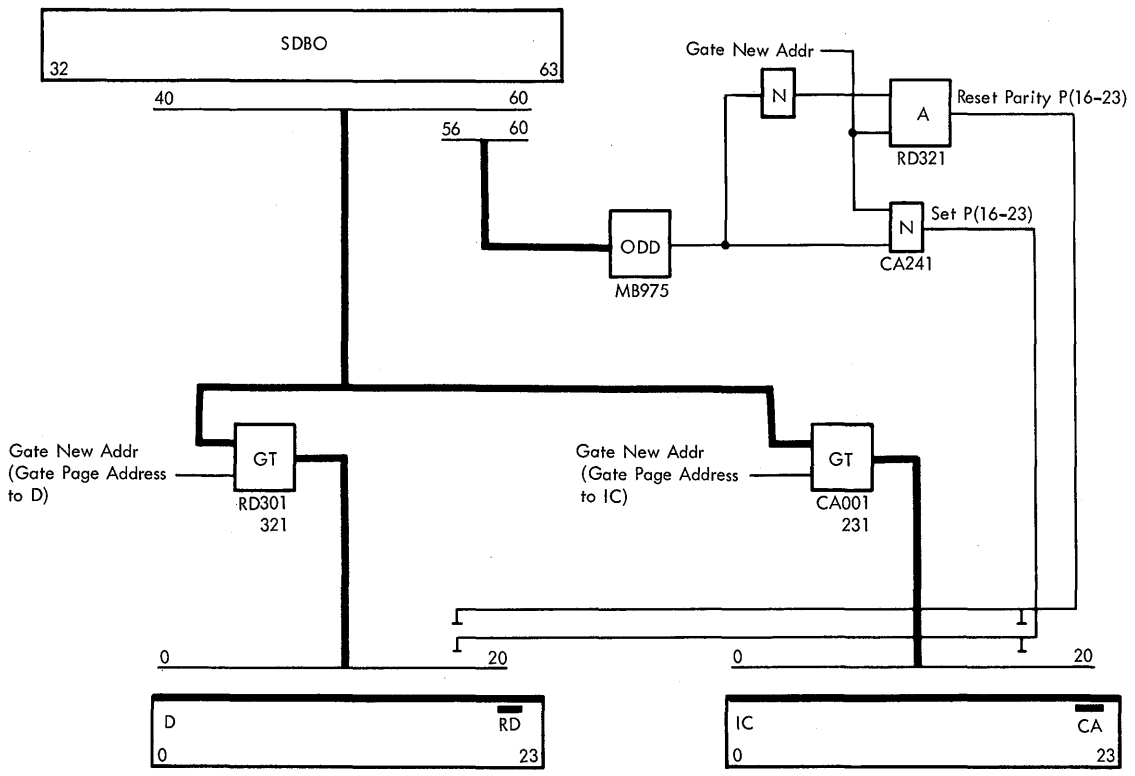


Figure 2-66. Page Gates for D and IC

check on the page address occurs if SDBO(61–63) is not equal to 0. This check sets the 'page exception' trigger. A 'page complete' signal is activated to reset the page control logic. SCI logic is reset in the normal manner.

Converting SAB Parity

- Generate parity bit for SAB (1–5).
- Generate parity bit P_A for SAB (6–12).
- Generate parity bit P_B for SAB (13–19).

The SAB parity conversion logic generates three parity bits: SAB P(1–5), P_A for SAB (6–12), and P_B for SAB (13–19). This logic takes into account the setting of the DEFEAT INTERLEAVING switch on the CE control panel. If this switch is not in the PROC position, it causes SCI logic to reverse SAB (6) and SAB (20), and this same bit reversal must be performed in the parity conversion logic for generating P_A and P_B parity bits.

The 'gate PSBAR to SAB' signal also affects the parity conversion logic because it gates logical and physical PSBAR to SAB when it is active or gates the contents of an ATR slot to SAB when it is not active. The same gating must be performed in the parity conversion logic.

The SAB parity conversion logic is shown in Diagram 4-610, FEMDM. This logic uses exclusive-OR circuits to generate, subtract, and add parity bits until the required results are obtained. Note that the output of an exclusive-OR always excludes those bits on the inputs if both inputs are active. Thus, depending on the input bits, an exclusive-OR can be used as an adder or subtracter.

When SAB P(0–3) is subtracted from SAB P(0–3, 6 or 20, 7) at an exclusive-OR, the result is SAB P(6 or 20, 7); i.e., SAB P(0–3) has been canceled. In this manner, SAB P(1–5), P_A and P_B are generated and sent to the SE/DE interface.

Resetting of SCI Logic

- 'Accept' signal from storage activates 'BCU cleanup' signal to reset SCI logic.

'BCU cleanup' logic and timing are shown in Figure 2-67. When an 'accept' pulse is received from the selected SE or DE, it sets the 'accept' trigger. The 'accept' trigger activates 'BCU cleanup' and starts a time delay. When the time delay times out, 'late BCU cleanup' is activated, and 'accept' trigger is reset. The 'late BCU cleanup for CPU requests' is activated by 'late BCU cleanup' at not clock time and is deactivated at the following not clock time. All SCI logic is reset by this action.

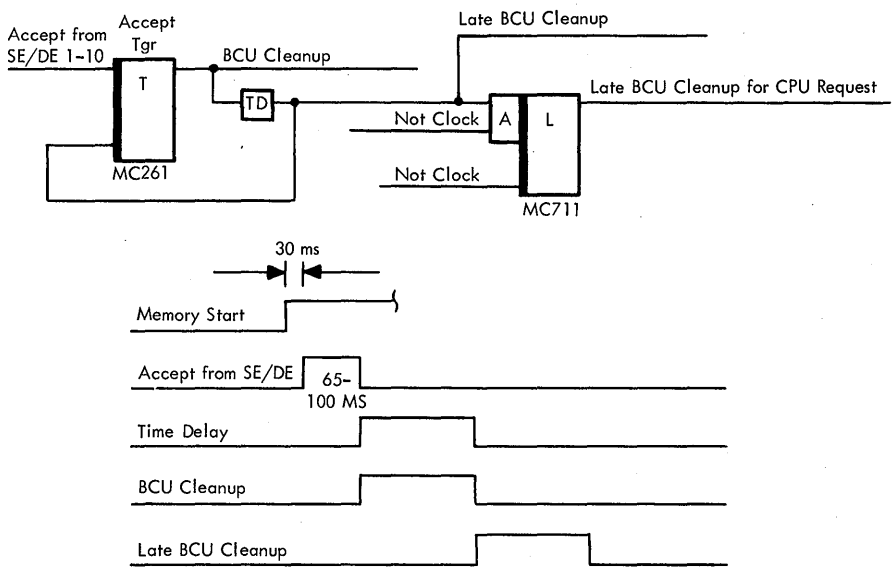


Figure 2-67. 'BCU Cleanup' Logic and Timing

DETAILED ANALYSIS OF SCI OPERATIONS

The subsequent paragraphs describe the operational sequences performed in the SCI during processing of CE storage requests. (See Diagram 4-611, FEMDM.)

CE storage requests are issued to the SCI from I-Fetch, ROS, and Scan logic, and both the storage request and the resulting data transfer are overlapped with processing. Although all CE requests are handled by the SCI as basic fetch or store requests, the following variations exist: three- and four-cycle fetch, store, insert-key, set-key, test-and-set, and single-cycle operation.

Three- and Four-Cycle Fetch Operation

- Ingating of requested storage data is specified at three or four cycles following storage request.

Because the SCI operates at the same machine cycle speed as the CE, and because the access time to storage requires three cycles, the CE is allowed to continue processing for three cycles following the request. The SCI must keep track of the CE cycle progression so that the SDBO ingating is executed at the correct time. The '3-cycle' trigger affects the length of time the CE clock will be stopped by conditioning the setting of CPU sequencer latches 'CPU 3' and 'CPU 4'. For detailed timing of three- and four-cycle fetch operations, refer to ALD A7501.

Store Operation

- Store-data requests are always made per D.

Data from ST or XY registers is gated to the SDBI for transfer to storage. Mark trigger settings are transferred to storage (via the 'mark' bus) to specify which of the eight bytes of data are to be stored. For detailed timing of the store operation, refer to ALD A7521.

Insert-Key Operation

- 'D-storage request' and 'insert key' signals are sent from ROS to SCI.
- Basic fetch operation is performed by SCI.
- Five-bit (plus parity) storage protection key is transferred from storage to CE via 'key out' bus.

Essentially, insert-key operations are fetch requests per D, in which a five-bit (plus parity) storage protection key is obtained from the specified storage protection area of main storage and inserted into F(0–4) of the CE. These operations enable the CE to examine the key patterns used by the storage protection mechanism.

An insert-key request sets the 'insert key' trigger in the SCI to modify the normal stepping of the 'CPU sequencers': the 'CPU 5' trigger is set on the first cycle

following the insert-key request, and further sequencer stepping is not performed. The 'stop CPU clock' trigger is also set on the first cycle following the insert-key request. Thus, the CE clock is stopped on the second cycle after the request.

The contents of D are gated to the SAB, and a 'select' signal is generated and sent to the addressed SE or DE together with an 'insert key' signal. In the SE or DE, the two signals ('select' and 'insert key') initiate an insert-key operation. SAB (6–13) is decoded to determine the protection key location, and the key pattern (five bits plus parity) is fetched from that location. The SE or DE then generates an 'advance key' signal, which prepares the SCI for ingating of the 'key out' bus into the F-register of the CE. A detailed timing chart for the insert-key operation is shown on ALD A7521.

Set-Key Operation

- 'D-storage request' and 'set key' signals are generated from ROS to SCI.
- Basic store operation is performed by SCI.
- Five-bit (plus parity) storage key is transferred from CE to storage via 'key in' bus.

Essentially, set-key operations are store requests per D in which a five-bit (plus parity) storage protection key is obtained from F(0–4) in the CE and stored into the specified storage protection area of an SE or DE. These operations enable the CE to set new key patterns into the storage protection mechanism.

A set-key request sets the 'set key' trigger in the SCI. In addition, all CE mark triggers are set during the set-key operation. Receipt of the D-storage request by the SCI then sets the 'store' trigger and starts the 'CPU sequencers' in the normal manner. The contents of D are gated to the SAB, and the 'select' and 'set key' signals are sent to the addressed SE or DE. (The sending of the 'store' signal to the SE or DE is inhibited during the set-key operation.)

The 'select' and 'set key' signals initiate a set-key operation in the storage unit. SAB (6–13) is decoded to determine the protection key location, and the contents of the 'key in' bus are gated into that location. For a detailed timing chart of the set-key operation, see ALD A7521.

Test-and-Set Operation

Although a test-and-set request combines aspects of both fetch and store operations, the basic handling of the request by the SCI is similar to a three-cycle fetch per D. The major difference is that the SCI sends one mark signal and a 'test and set' signal to the SE or DE specified by the D-address.

Single-Cycle Operation

- START pushbutton provides for manual stepping through CE cycle.
- RATE switch in SINGLE CYCLE STORAGE INHIBIT position provides for manual stepping through all cycles of the request sequence. (Storage unit is not selected; data transfer is inhibited.)
- RATE switch in SINGLE CYCLE position enables CE to run automatically from the time 'select' signal is sent to storage until data transfer operation is completed.

When CE operations are being tested in the single-cycle mode, the CE clock is stepped manually; one CE clock cycle results for each depression of the START pushbutton. The SCI clock, however, is not affected by the single-cycle mode and runs automatically, thus allowing the SCI to continue servicing storage requests from the CE. The servicing of storage requests during single-cycle mode is shown in Diagram 4-612, FEMDM.

The single-cycle operation can be performed by the CE with or without access to storage. If the RATE switch is placed in SINGLE CYCLE position, the CE will access storage whenever it steps through a cycle specifying a storage request; if the RATE switch is in SINGLE CYCLE STORAGE INHIBIT position, all storage requests are ignored by the SCI.

When servicing storage requests from the CE in the single-cycle mode, the SCI must ensure that the gating of data to or from the CE is synchronized with the storage unit operation. To accomplish this function, special single-cycle logic in the SCI controls the CE clock and runs it automatically whenever synchronized ingating is required.

To enable manual stepping through as many CE cycles as possible, the SCI delays sending the 'select' signal to storage until the 'CPU sequencers' stop the CE clock. The stop-clock condition indicates to the SCI that the data transfer between CE and storage must be executed on the next depression of the START pushbutton. At this point, the nature of the request is of primary consideration. If a fetch-data request is in progress, the SCI must override the single-cycle controls and run the CE clock automatically until the CE executes the ROS word with the 'ingate SDBO' micro-order. If a store-data request is in progress, the SCI need not control the CE clock because the CE data is placed on the SDBI when a 'select' signal is sent to

storage, i.e., as soon as the START pushbutton is depressed. Thus, on store-type requests, the operator can single-cycle through every ROS word of the CE microprogram; on fetch-type requests, the operation automatically skips over one or two ROS words, depending on whether a three- or four-cycle request is specified. (Note that the time slice between two consecutive depressions of the START pushbutton does not enter into consideration; this time slice is much greater than the 600-ns time interval required to access storage.)

When START is depressed and an 'accept' signal is received from storage, the resulting 'BCU cleanup' signal restarts the CE clock. The state of the 'CPU sequencers' after the first CE clock signal is generated indicates the type of request in progress and whether additional stepping of the CE clock is required; this stepping is performed automatically under control of the 'CPU clock go' trigger. Note that the 'CPU clock go' trigger is always reset on the first clock signal after the 'CPU 5' latch is set.

If a store-data request is in progress, the 'CPU 5' latch is set before the CE clock is restarted. Thus, as soon as the CE clock is restarted, the 'CPU clock go' trigger is reset to indicate that no additional CE clock cycles are required to complete the request.

If a three-cycle fetch-data request is in progress, the 'CPU 4' latch is set before the CE clock is restarted. When the CE clock is restarted, the first clock signal sets the 'CPU 5' trigger/latch sequencers and accesses the ROS word with the 'ingate SDBO' micro-order. Note, however, that the ingating of the SDBO into the CE takes place on the following cycle. Thus, the CE clock must be automatically stepped an additional cycle to perform the ingating. This function is performed by the 'CPU clock go' trigger, which is reset by the same clock signal that gates the data into the CE.

If a four-cycle fetch-data request is in progress, the 'CPU 3' latch is set before the CE clock is restarted. In this case, the 'CPU clock go' trigger is not reset until two cycles after clock-restart. Thus, the CE clock is automatically stepped through two additional cycles to perform the required ingating.

As mentioned previously, sending the 'select' signal to storage is delayed when servicing requests in the single-cycle mode. This delay is accomplished as shown in Diagram 4-612, FEMDM.

When servicing requests in the single-cycle mode, the 'CPU clock go' trigger is set and remains set until the request is completed. The 'request finishing' trigger prevents sending another 'select' signal if another sequential CE request (in single-cycle mode) has been entered into the sync latch. After the current request has been serviced, the 'CPU clock go' and 'request finishing' triggers are both reset.

This chapter, which discusses the 7201-02 CE instructions, is divided into ten sections:

Section 1, Instruction Fetching.
Section 2, Fixed-Point Instructions.
Section 3, Floating-Point Instructions.
Section 4, Decimal Instructions.
Section 5, Logical Instructions.
Section 6, Branching Instructions.
Section 7, I/O Instructions.
Section 8, Status Switching Instructions.
Section 9, Multiple Computing Element Instructions.
Section 10, Display Instructions.

Machine operation during instruction fetching and execution is controlled by ROS microprograms which are represented by CLD's. The discussions in the following sections are based upon simplified versions of the CLD's and upon upper-level, positive-logic diagrams located in the associated FEMDM.

SECTION 1. INSTRUCTION FETCHING

Basic control for the instruction fetching (I-Fetch) operation is derived from one of four possible microprograms, depending on the format of the instruction being fetched. Each microprogram performs routines dictated by the instruction format (RR, RX, RS and SI, or SS) and is therefore common to many instructions. (The same microprogram governs the I-Fetch of RS and SI Instructions.) Subsequently, a branch is made to an appropriate microprogram for execution of a specific instruction. These individual execution sequences all terminate with a branch back to the I-Fetch microprogram to continue the sequence.

A typical microprogram sequence is shown in Figure 3-1. The correct I-Fetch microprogram to be entered upon completion of an instruction is dependent on the format of the instruction to be executed next. A test for the format of the upcoming instruction is made on the last cycle of the execution phase. The various actions performed during this last cycle (called the end operation or end-op cycle) must be thoroughly understood before undertaking a detailed analysis of each I-Fetch sequence.

BASIC END-OP CYCLE

- End-op cycle completes execution of instruction and initiates fetching of next instruction.
- End-op cycle is governed by normal end-op or branch end-op ROS word.
- Branch end op is used to speed execution of branch-type operations.

The end-op cycle is the last cycle in the execution phase. During this cycle, actions dictated by the execution phase of the instruction are completed and the fetching of the

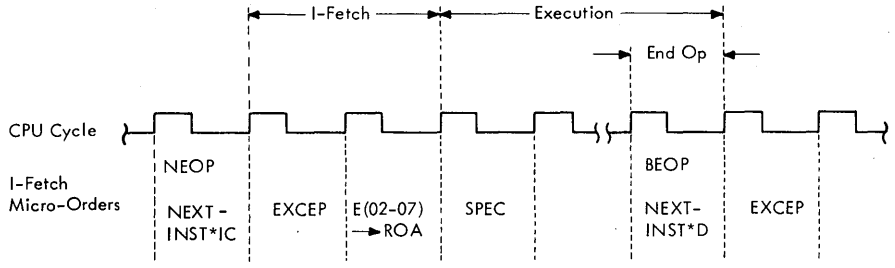


Figure 3-1. Typical Microprogram Sequence

next instruction begins. The execution phase is completed by setting the CC (if specified in the instruction) and by detecting interruptions or exceptional conditions that may have occurred during the execution phase. (The recovery microprograms are discussed after the basic end-op and I-Fetch sequences.)

The instruction fetching begins by:

1. Decoding the format of the upcoming instruction.
2. Initiating the operand fetch required by that format.
3. Establishing the correct I-Fetch sequence which is to follow.
4. Detecting the need for more instructions, and requesting new instructions from main storage when the need exists.

This discussion deals with those end-op actions that affect the subsequent I-Fetch sequence. Although instruction fetching begins during the end-op cycle, the next cycle is defined as the first I-Fetch cycle.

The setting of the CC affects the subsequent I-Fetch only when the upcoming instruction is a Branch on Condition instruction. Depending on the CC, new instructions may be requested from D (condition met) or from the IC (condition not met). The manner in which the CC is set is discussed in the specific execution sequences described in this chapter (Sections 2 through 10).

The actions performed during the end-op cycle are governed by two basic ROS end-op words: normal end-op and branch end-op. (Although they perform different functions during end-op, they perform the same functions for the subsequent I-Fetch sequence.) The normal end-op word is in control of the end operation if the address of the next instruction is specified by the IC. The next instruction is decoded from R. Conversely, the branch end-op word is in control if the address of the next instruction is specified by D. In this case, the next instruction is decoded from the SDBO (the effective R) at the start of the end operation.

The primary function of the branch end-op word is to fulfill specific timing requirements imposed upon execution of some branch instructions (see Section 6 of this chapter). Two conditions lead to a branch end-op micro-order:

1. Sometimes upon execution of a successful branch, end-op takes place before the address of new instructions (in D) has been transferred to the IC. In such cases, the branch end-op word is always in control. To establish the correct I-Fetch microprogram for the next instruction, the branch end-op word samples D(21,22) and the effective-R(0,1) bits; i.e., bits 0 and 1 of the op-code halfword to be transferred to R are sampled directly from the SDBO. Thus, the I-Fetch microprogram for the next instruction is established as soon as the instructions (specified by the branch) arrive from main storage.
2. Except for the Branch on Condition instructions, the Element assumes that all branches are successful. Accordingly, upon predecoding a branch instruction, the Element inhibits any IC request to refill Q and, instead, requests instructions per the branch address (in D). If, during execution, the branch proves to be unsuccessful, the instructions accessed by the D-request are not gated into Q, and the Element must resume processing of the instructions specified by the IC. At this time it may be found that the unsuccessful branch was the last instruction in Q. Although a request per the IC is immediately generated, at least three CE cycles must elapse before the CE can resume normal processing. Also, because the format of the instruction is usually decoded from R(0,1), additional time would be lost if the first halfword (arriving from main storage) had to be gated to R before the I-Fetch microprogram for the instruction could be established. Under such conditions, use of the branch end-op word increases the speed in establishing the I-Fetch microprogram for the next instruction. The instruction address (in the IC) is temporarily transferred to D. Instead of sampling R(0,1) the branch end-op word samples the effective-R(0,1) to establish the correct I-Fetch microprogram immediately upon arrival of the instructions from main storage.

Prefetching of Operands During End Op

- For RR instructions, one LS register is accessed by R1 field if not a branch instruction; by R2 field if branch instruction.
- For RX, RS, SI, and SS instructions, one LS register is accessed by B-field.

During the end-op cycle, R contains the op-code halfword of the next instruction. The format of the instruction is established by sampling R(0,1), and the operand prefetch dictated by that format is initiated. The end-op cycle is completed with the 'R→E' micro-order, which transfers the op-code halfword to E at the start of the I-Fetch sequence.

The scheme for prefetching operands during end-op time is shown in Diagram 5-1, FEMDM. During this time, an LS register specified in the R or B field of the upcoming instruction is addressed and transferred to T. The desired LS register is addressed by gating the appropriate field of the instruction to LAL. Ingating to LAL is initiated by the 'NEOP' micro-order in the normal end-op word or by the 'BEOP' micro-order in the branch end-op word.

The format of the upcoming instruction is established by decoding R(0,1):

R(0,1)	Instruction Format
00	RR
01	RX
10	RS or SI
11	SS

When an RR format is decoded, a further test is performed to determine whether the upcoming instruction is a branch. If the instruction is not a branch, the R1 field [R(8–11)] is gated to LAL. For an RR branch, however, the R2 field [R(12–15)] is gated to LAL. This action is necessary because, for branch instructions, R2 specifies the LS register containing the branch address. Since in this case a storage request for new instructions must be made as soon as possible, R2 must be gated to LAL first.

When an RX, RS, SI, or SS format is decoded, a test is made to determine which of the four halfword positions in Q contains the second halfword of the upcoming instruction. The B-field of the selected halfword is then always gated to LAL. Selection of the correct halfword in Q depends upon the ROS word (branch or normal) in control of the end-op. The normal end-op word specifies that the address of the upcoming instruction is contained in the IC. In this case, IC(21,22) indicates the Q portion from which the first halfword of the instruction has been transferred to R. Consequently, these bits are decoded to select the second halfword of the instruction in Q. The branch end-op word is in control when the address of the upcoming instructions is in D. Because in this case D(21,22) points to the correct Q position, these bits are used to select the correct B-field in Q.

Prefetching of operands from LS is from GPRs 0–15 (decimal), unless an RR format, floating-point instruction has been predecoded. In this case, the FPR addressed by R1 is selected by forcing LAL(0) to 1. The contents of the LS register accessed during the end-op cycle are always transferred to T. This action is performed by the '→T' micro-order in the end-op word. Thus, at the start of an I-Fetch sequence, T always contains an operand (per R-field) or the base portion of an operand address (per B-field).

At the completion of an end-op cycle, the halfword containing the op code of the instruction is transferred to E (initiated by the 'R→E' micro-order in the end-op word). Thus, further operand prefetching (by the subsequent I-Fetch sequence) is performed with the op code in E.

Fetching of Instructions by End-Op Micro-Order

A test to establish whether new instructions are required is always performed during end op. If the upcoming instruction is not a branch and Q needs to be refilled, a request for new instructions is generated at end op. If the upcoming instruction is a branch, the storage request is blocked during end op.

Under certain conditions, it is possible to request new instructions from main storage one or two cycles before end-op. This action is initiated by the 'early end-op' (EEOP) micro-order, contained in the execution sequences of some instructions. All execution sequences, including those with the 'EEOP' micro-order, terminate with the end-op word.

A Q-register refill exceptional condition usually follows an end-op request for new instructions. This exceptional condition adds one cycle to the basic RR, RX and RS, and SI I-Fetch routines.

Requests During End Op

During the end-op cycle, a test is made to establish whether Q needs to be refilled with new instructions. The outcome of this test depends upon the format of the upcoming instruction, on its position in Q, and on whether it is a branch or the subject instruction of an Execute instruction.

As shown in Diagram 5-2, FEMDM, a test of the status of Q is initiated by the normal end-op (NEOP) or branch end-op (BEOP) micro-order contained in the normal or branch end-op word, respectively. Upon the decoding of the 'NEOP' micro-order, IC(21,22) is sampled to establish which halfword position in Q has been transferred to R. The same function is performed by the 'BEOP' micro-order when the address of the upcoming instruction is contained in D. In this case, D(21,22) is examined to establish which halfword in Q is to be processed next. Depending on the instruction format decoded from R(0,1), and if the upcoming instruction is neither a branch nor the subject of an Execute instruction, storage requests per the IC may be generated when the first, second, or third halfword position in Q is to be processed next.

Q-Position Transferred to R	Setting of IC(21,22) or D(21,22)	Instruction Format	Type of Request
1st	00	SS	4-cycle
2nd	01	SS	4-cycle
		RX, RS, or SI	3-cycle
3rd	10	All formats	3-cycle
4th	11	All formats	None

Q has already been refilled during the instruction being completed if bits 21 and 22 = 11; therefore, another refilling of Q is not necessary.

Requests During Early End Op

Execution sequences of some instructions contain the 'EEOP' micro-order. The function of this micro-order, which is given 1 or 2 cycles before the 'NEOP' micro-order, is to examine the instruction status in Q and to initiate an early storage request if Q needs refilling. Requests initiated

by the 'EEOP' micro-order are blocked if the next instruction to be executed is (1) a branch instruction, (2) an SS instruction, or (3) a subject of an Execute instruction.

Early requests to refill Q are generated according to conditions shown in Diagram 5-3, FEMDM. The normal end-op request is blocked when an early request is in progress. Note that the 'EEOP' micro-order can only initiate a 4-cycle request. The advantage of an early request is that the SCI will address main storage 1 or 2 cycles before end op. When initiated 2 cycles before end op, the refilling of Q does not force the Q-register refill exceptional condition if the instruction being fetched is of the RR or indexed RX format or is a shift instruction.

Selection of I-Fetch Microprogram

- Selection of I-Fetch sequence is controlled by 'NEXT-INST*IC' micro-order during normal end op or by 'NEXT-INST*D' micro-order during branch end op.
- 'NEXT-INST*IC' micro-order specifies functional ROS branch per R(0,1), IC(21,22), B = 0, and X2 = 0.
- 'NEXT-INST*D' micro-order specifies functional ROS branch per effective-R(0,1), D(21,22), B = 0, and X2 = 0.

The correct I-Fetch sequence is entered by establishing the address of the first ROS word in that sequence. This address is then placed into ROSAR so that the desired ROS control word may be obtained on the following cycle.

ROSAR(0–5) is furnished directly by the end-op word as 001000. These bits designate the address of a general I-Fetch operation about to take place. To arrive at the specific I-Fetch sequence (RR, RX, RS and SI, or SS), the bit configuration of ROSAR(6–11) must be established. The manner in which ROSAR(6–11) is established is determined by the ROS word (normal or branch) in control of the end op.

The normal end-op word contains the 'NEXT-INST*IC' micro-order specifying a 64-way functional branch. This micro-order sets ROSAR(6–11) according to the following conditions:

ROSAR Bit	Condition
6	Set if R(0) = 1
7	Set if R(1) = 1
8	Set if instruction X2 field = 0, and RX format
9	Set if instruction B field = 0, and not RR format
10	Set if IC(21) = 1
11	Set if IC(22) = 1

The above actions specify the format of the upcoming instruction, the type of further operand fetch required, and the number of counts by which IC(21,22) must be increased to select the first halfword of the instruction following it in main storage.

The registers affected by the 'NEXT-INST*IC' micro-order are shown in Diagram 5-5, FEMDM. The format of the upcoming instruction is decoded from R(0,1). For non-RR instructions, a test is made to determine whether the B-field of the instruction is equal to zero and, in the case of RX instructions, whether the X2 field is also zero.

The zero test for the B and X2 fields is necessary to establish a correct address computation by the subsequent I-Fetch routine. To increase the speed of operand prefetching, the B-field is always gated to LAL during the end-op cycle. A zero address to LAL accesses LS register 0, the contents of which may not necessarily be zero. However, the condition of B-field being zero requires that the base portion of the operand address be zero. Thus, the subsequent I-Fetch sequence selected must ignore the contents of LS register 0 (accessed by a zero B field). Similarly, in the case of the X2 field being zero, the I-Fetch sequence selected must not address LS per the X2 field. The manner in which the correct I-Fetch sequence is selected is described below.

Four 4-way AND's simultaneously sample the four possible B-field locations in Q. Each AND is conditioned if its corresponding four-bit input consists of all zeros. As explained previously, IC(21,22) selects the first halfword of the instruction that has been transferred from Q to R. Therefore, these bits are used as gates to select the second halfword of the instruction in Q. If, for example, the first halfword position of Q has been transferred to R and decoding of R(0,1) shows that the instruction is not of the RR format, Q(16–19) must be selected to obtain the correct B-field. When the first halfword position in Q is transferred to R, IC(21,22) is set to 00. This setting (coupled with the absence of an 'RR block' signal) selects the output of the AND that samples the correct B-field; i.e., Q(16–19). When the B-field of the instruction is found to be zero, ROSAR(9) is set to 1. This action addresses an I-Fetch microprogram that ignores the contents of the LS register accessed by the B-field.

When the upcoming instruction is of an RX format, a similar test is performed to establish whether the X2 field of the instruction, R(12–15), is equal to zero. Upon detecting a zero X2 field, ROSAR(8) is set to 1. This action dictates that the subsequent I-Fetch microprogram does not address LS per X2; i.e., E(12–15).

The ROS branch specified by the 'NEXT-INST*IC' micro-order is completed by forcing IC(21,22) into ROSAR(10,11). This action allows the first I-Fetch microinstruction to correctly update IC(21,22) and R without further testing.

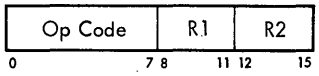
The 'NEXT-INST*D' micro-order in the branch end-op word sets ROSAR(6–11) according to the following conditions:

ROSAR Bit	Condition
6	Set if effective - R(0) = 1
7	Set if effective - R(1) = 1
8	Set if instruction X2 field = 0, and RX format
9	Set if instruction B field = 0, and not RR format
10	Set if D(21) = 1
11	Set if D(22) = 1

Note that ROSAR(6,7) is set from the effective-R rather than from R, and that ROSAR(10,11) is set from D(21,22) rather than from IC(21,22). This is done because R and IC are either still invalid or are just being set by the branch operation in progress.

BASIC RR I-FETCH

- RR format:



- Purpose:
  1. For nonbranch instructions, load 1st operand into A, B, and D. Load 2nd operand into S and T.
  2. For branch instructions, load 2nd operand into A, B, and D. Load 1st operand into S and T. Request new instructions, if needed.
  3. Set STC to 100 and ABC to 000.
- Conditions at start of I-Fetch:
  1. Instruction is transferred to E.
  2. If instruction is not a branch, 1st operand is in T; for a branch, 2nd operand is in T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of RR instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The RR instructions basically require a 1-cycle I-Fetch. The actions initiated during this cycle are governed by 1 of 4 possible ROS control words selected at end-op time. Selection of the specific ROS word depends on the original position of the RR instruction in Q. This word contains the appropriate micro-orders for incrementing IC(21,22) and for transferring the first halfword of the next instruction to R. Except for these actions, the functions performed by the four ROS words are identical. Diagram 5-6, FEMDM, is a simplified flowchart of an RR I-Fetch; Diagram 5-7 shows the data registers used.

If no interruption or exceptional condition is detected, the entire RR instruction is transferred to E at the start of the I-Fetch cycle (by the 'R→E' micro-order at end op). The operand prefetching, initiated at end op, is then continued. The order in which operands are prefetched depends on whether the instruction is a branch:

1. For nonbranch instructions, the first operand (accessed during end op) is transferred from T via the parallel adder to A, B, and D. The second operand is then addressed by gating E(12–15) to LAL. When the second operand is accessed, it is loaded into S and T.
2. The above order is reversed for branch instructions; i.e., the second operand (accessed during end op) is placed into A, B, and D while the first operand is placed into S and T. A storage request per the branch address is generated subject to the conditions shown in Diagram 5-6.

The correct execution sequence is entered by establishing the address of the first ROS word in that sequence. This address is determined by sampling the instruction op code from E(2–7) by means of the 'E(02–07)→ROA' micro-order. As stated earlier, this description of RR I-Fetch applies only when no exceptional conditions or interruptions are present. The ROS word governing the I-Fetch cycle always contains the 'EXCEP' micro-order, which can override the functional branch per the instruction op code. Therefore, the branch to the first execution cycle occurs only when there are no interruptions or exceptional conditions.

In addition to prefetching the operands, the I-Fetch ROS word contains appropriate micro-orders to increment IC(21,22) and to transfer the first halfword of the next instruction to R. IC(21,22) is set one count higher ('X→IC' micro-order) to point at the next instruction. The first halfword of the next instruction is transferred to R by the 'QXX→R' micro-order.

Included in the first RR I-Fetch word is the 'RESET' micro-order, which causes the following actions:

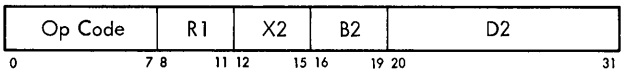
1. During I-Fetch of branch instructions, initiates the request for new instructions (see Diagram 5-4) and gates E(8–11), instead of E(12–15), to LAL.
2. Resets all STAT's and Edit-instruction controls.
3. Sets STC to 100 and ABC to 000.
4. Forces LAL(0) to 1 for floating-point instructions, causing the FPR's to be addressed.
5. Sets the 'stop' trigger if operating at the instruction-step rate.

Note that IC(21,22) is not advanced, the Q-to-R transfer is not effected, and unsuccessful branch-on-condition requests are not generated if the 'execute in progress' trigger is set. The set state of this trigger indicates that the current I-Fetch is for a subject instruction of an Execute

instruction. Therefore, the address of the RR instruction is specified by D and not by the IC.

BASIC RX I-FETCH

- RX format:



- Purpose:
  1. Compute address of 2nd operand and transfer to D; request 2nd operand from main storage, if necessary.
  2. Transfer 1st operand to S and T.
- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferred to E; 2nd halfword is in Q.
  2. Contents of LS register specified by B2 are transferred to T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of an RX instruction. It is assumed that no interruption or exceptional conditions were detected in the preceding end-op cycle.

The RX instructions basically require a 1- or 2-cycle I-Fetch. The actions initiated during the first I-Fetch cycle are governed by 1 of 16 possible ROS control words selected at end-op time. This selection depends on whether the B2 and/or X2 fields of the instruction are zero, and on the original position of the RX instruction in Q. The zero test establishes four separate cases for the I-Fetch routine: (1) B2 = 0 and X2 = 0, (2) B2 ≠ 0 and X2 = 0, (3) B2 = 0 and X2 ≠ 0, (4) B2 ≠ 0 and X2 ≠ 0. The first two cases require a 1-cycle I-Fetch; the last two, a 2-cycle I-Fetch. Each of the above four routines contains appropriate micro-orders for incrementing IC(21,22) and transferring the first halfword of the next instruction to R. Consequently, a four-way branch is inherent in each routine, depending on the previous IC(21,22) setting; i.e., 00, 01, 10, or 11.

Diagram 5-9, FEMDM, is a simplified flowchart of an RX I-Fetch; Diagram 5-10 shows the data registers used. If no interruptions or exceptional conditions are detected, the op-code halfword of the RX instruction is transferred to E at the start of I-Fetch (initiated by the 'R→E' micro-order at end op). The operand prefetch routine is then continued with the first halfword of the instruction in E and the second halfword in Q. The I-Fetch of non-indexed RX instructions (X2 = 0) is described first.

The 'X→IC' micro-order issued by the first I-Fetch word sets IC(21,22) two counts higher. The first halfword of the next instruction is transferred to R by the 'QXX→R' micro-order if it is now in Q; that is, if IC(21,22) did not equal 10 at the start of I–Fetch. If the instruction is not indexed, the address of the second operand is obtained by adding D2 to the base address. The contents of the LS register per B2 are placed into T at the start of I-Fetch. If B2 was found to be zero during end op, the contents of T are ignored, and the appropriate D2 field in Q is selected and routed to D via the parallel adder. However, if B2 ≠ 0, the contents of T and the D2 field are gated simultaneously to the parallel adder, and the resultant sum is transferred to D. A 3-cycle storage request for the second operand is made from D if the following conditions do not exist:

1. A Q-register refill exceptional condition is in progress.
2. The instruction is Store Halfword, Store Character, or Load Address.
3. The instruction is an unsuccessful Branch on Condition. (A request for new instructions is issued from the IC, if necessary.)

The first operand is obtained from LS per R1 and transferred to S and T. At the completion of the I-Fetch cycle, a branch is made to a specific execution sequence as determined by the 'E(02–07)→ROA' micro-order.

In the case of indexed RX instructions (X2 ≠ 0), two cycles are required to complete the I-Fetch routine (Diagram 5-11, FEMDM). During the first cycle, D2 is added to the contents of T (if B2 ≠ 0) and the result is temporarily stored into B. The LS register specified by X2 is then accessed, and its contents are placed into T. The contents of T and B are added during the second cycle, and the sum (second operand address) is transferred to D. The conditional storage request is now made. After the first operand is obtained from LS and placed into S and T, a branch per the instruction op-code is made to enter the correct execution sequence.

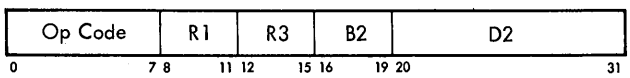
The 'RESET' micro-order:

1. Resets all STAT's and Edit instruction controls.
2. Resets STC and ABC to 000.
3. Initiates any necessary storage requests for branch instructions and for subject instructions of the Execute instruction.
4. Forces LAL(0) to 1 for floating-point instructions, causing the FPR's to be addressed.
5. Sets the 'stop' trigger if operating at the instruction-step rate.

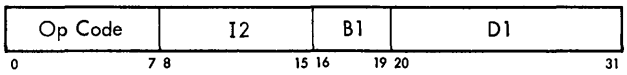
If the instruction being fetched is the subject of an Execute instruction ('execute in progress' trigger is set), the incrementing of IC, the Q-to-R transfer, and the unsuccessful Branch on Condition requests to refill Q are inhibited.

BASIC RS AND SI I-FETCH

- RS format:



- SI format:



- Purpose:
  1. Add contents of LS register specified by B-field to D-field; place result into D.
  2. Request operand from main storage, if necessary.
  3. For RS instructions, load 1st operand into S and T. (Contents of S and T are ignored for SI instructions.)
- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferred to E; 2nd halfword is in Q.
  2. Contents of LS register specified by B-field are transferred to T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of RS and SI instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The RS and SI instructions basically require a 1-cycle I-Fetch. The actions initiated during this cycle are governed by 1 of 8 possible ROS control words selected at end-op time. This selection depends on whether the B-field of the instruction is zero, and on the original position of the instruction in Q. The zero test establishes two distinct I-Fetch routines: (1) B = 0 and (2) B $\neq$ 0. A four-way branch is inherent in each routine, depending on the previous IC(21,22) setting; i.e., 00, 01, 10, or 11.

Diagram 5-13, FEMDM, is a simplified flowchart of RS and SI I-Fetch. If no interruptions or exceptional conditions are detected, the halfword containing the op-code of the RS or SI instruction is transferred to E at the start of I-Fetch (initiated by the 'R→E' micro-order at end op). The operand prefetch routine is then continued with the first halfword of the instruction in E and the second halfword in Q.

The LS register specified by the B-field (B1 or B2) is accessed during end op, and its contents are placed into T at the start of I-Fetch. If the B-field was found to be zero during end op, the I-Fetch routine ignores the contents of T, selects the appropriate D-field (D1 or D2) in Q and routes it to D via the parallel adder. If the B-field is not zero, the contents of T and the D-field are gated simultaneously to the parallel adder and the sum is then transferred to D. A 3-cycle storage request for the second operand is then made from D if the following conditions do not exist:

1. A request to refill Q was generated during the previous execution segment; i.e., IC(21,22) = 01 or 10. For this case, the ROS micro-order is not contained in the I-Fetch word.
2. The E-register contains a shift, Store Multiple, Move (MVI), Test and Set, or I/O instruction. For this case, the 'D sync' latch is prevented from being set.
3. The E-register contains a branch-on-index (BXH, BXLE) instruction. For this case, the 'RESET' micro-order resets the '3-cycle request' trigger, causing a 4-cycle storage request to be made from D regardless of IC(21,22).

Upon loading D with the second operand address, the I-Fetch routine proceeds to set IC(21,22) two counts higher, to transfer the first halfword of the next instruction to R if it is in Q, and to establish the first ROS control word for the execution phase. Fetching of the first operand per E(8–11) is meaningful only for RS instructions. For SI instructions, E(8–11) contains a portion of the immediate operand. Since a common ROS control word governs the I-Fetch of both formats, E(8–11) is always gated to LAL; the contents of the LS register thus accessed are placed into S and T. However, the subsequent execution sequences for SI instructions ignore the contents of S and T.

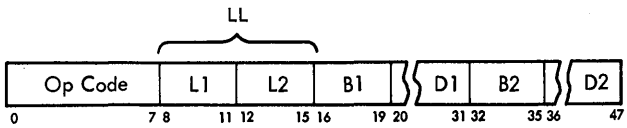
In addition to causing a 4-cycle storage request during the I-Fetch of a branch-on-index instruction, the 'RESET' micro-order:

1. Resets all STAT's and Edit-instruction controls.
2. Resets STC and ABC to 000.
3. Sets the 'stop' trigger if operating at the instruction-step rate.

If the instruction being fetched is the subject of an Execute instruction ('execute in progress' trigger is set), the incrementing of IC and the Q-to-R transfer is inhibited.

BASIC SS I-FETCH

- SS format:



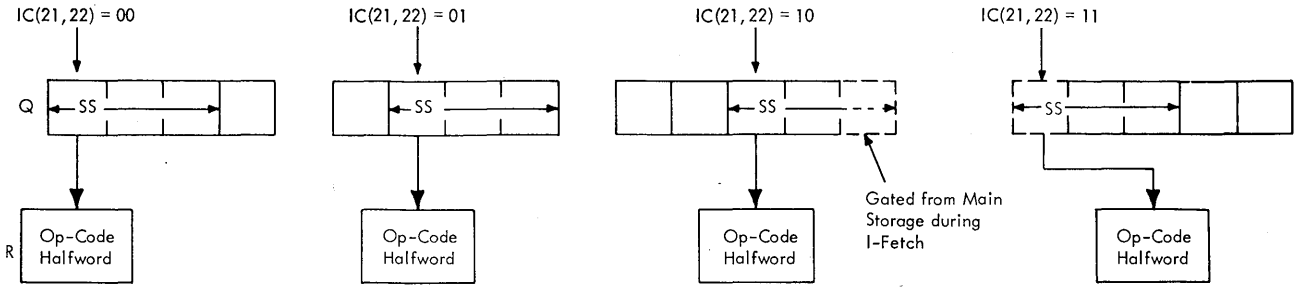
- Purpose:
  1. Transfer op-code halfword of next instruction to R; update IC and place into LSWR.
  2. Transfer computed address of 1st operand (destination) per instruction class to D; request destination operand from main storage (gated into CPU at start of 2nd execution cycle).
     a. Lowest destination address for logical instructions = base address (per B1) + D1.
     b. Highest destination address for decimal instructions = base address (per B1) + D1 + L1.

3. Transfer computed address of 2nd operand (source) to IC and T. Lowest source address = base address (per B2) + D2.
4. Perform ASC test (and invalid instruction address test if complete instruction is in Q).

- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferring to E, 2nd halfword is in Q, 3rd halfword is in Q if IC(21,22) ≠ 10 (otherwise 3rd halfword is gated to Q during 4th cycle of I-Fetch).
  2. Base address (per B1) is in T.
  3. Q refill is not in progress if IC(21,22) = 11.

The I-Fetch of SS instructions differs considerably from the I-Fetch routines described thus far (RR, RX, RS, and SI). Differences arise from three characteristics of the SS format: (1) the SS format is three halfwords long, (2) an SS instruction always stores the results into main storage, and (3) two main storage addresses are specified.

As previously stated, requests to refill Q are generated before the CE runs out of instructions. In describing the I-Fetch microprograms used for RR, RX, RS, and SI formats, it was assumed that the instruction to be executed was contained in Q. Because of the manner in which storage requests for instructions are generated, the assumption is valid for all 1- and 2-halfword instructions. For SS instructions, however, the I-Fetch routine may sometimes begin while the last halfword of the instruction is still in main storage. Figure 3-2 shows all possible locations that the SS instruction may assume in Q and the manner in which storage requests are generated for more instructions. Storage requests for SS instructions are generated (at end-op time) when IC(21,22) is set to 00, 01, or 10. If IC(21,22) = 00 or 01, the entire instruction is in Q at the start of I-Fetch. However, if IC(21,22) = 10, the last halfword of the instruction will arrive from main storage on the fourth cycle of I-Fetch. Consequently, processing of the



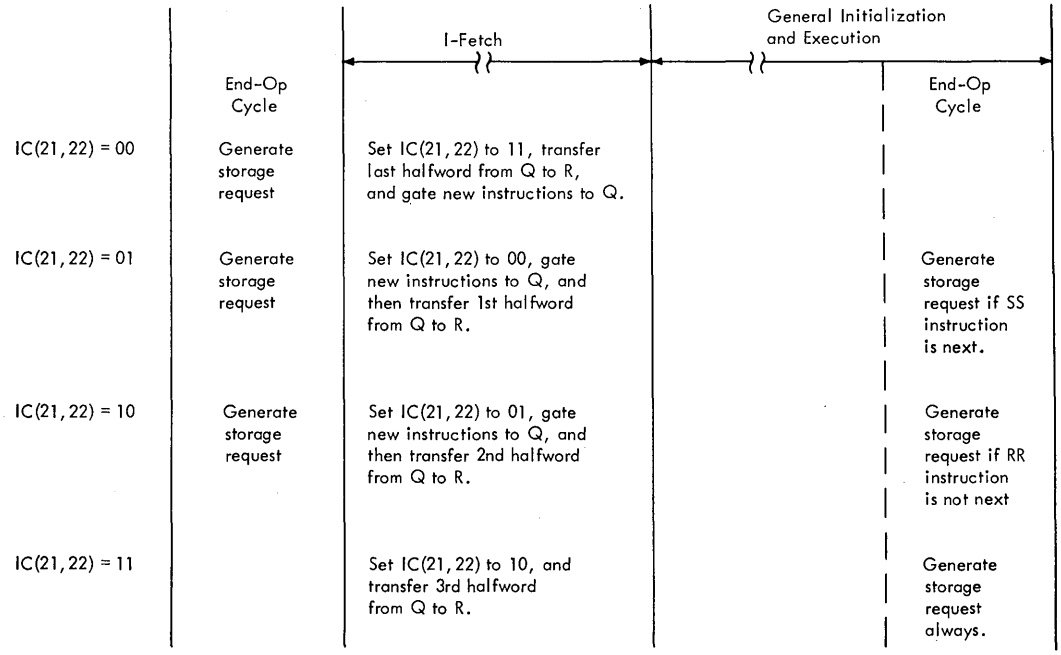


Figure 3-2. Basic Sequencing for SS Instructions

third halfword of the instruction cannot start until Q is refilled. Finally, if IC(21,22) = 11, at end op, Q has been refilled as a result of a previous end-op cycle and Q(0–31) contains the balance of the upcoming SS instruction.

An SS instruction operates on two operands obtained from main storage and stores the result into the same location from which the first operand was obtained. Therefore, the address of the first operand is also the destination address; the address of the second operand is commonly referred to as the source address. The first and second operand addresses are calculated in a manner similar to that of two-halfword instructions. The address of the first operand is computed first and loaded into D, and a storage request for the operand is made. The partial address of the second operand is then computed while the contents of the IC are transferred to the LSWR. The partial second operand address is loaded into the IC. After completing the I-Fetch routine, a General Initialization Sequence, GIS, is performed, after which control is transferred to the execution phase. (During GIS, the calculation of the second operand address is completed, and a storage request issued, if necessary.) Upon execution of the instruction, results are stored into main storage per the address in D (first operand or destination address).

Address Store Compare (ASC) Test

- Main storage address where data is to be stored is compared with address of current instructions.
- Comparison is made whenever data is stored into main storage.
- If data is stored at instruction address, 'PSC' trigger is set to indicate that instructions in Q must be refetched.
- For SS instructions, ASC test is performed during I-Fetch. Lower and upper limits of destination address are compared with instruction address.

An ASC test must be made each time the CE stores data into main storage. This test compares the destination address of the data with the current instruction address. If it is found that both addresses are the same, the 'program store compare' (PSC) trigger is set, indicating a need to refetch instructions; i.e., the instructions currently in Q must again be obtained from main storage because the next instruction to be executed may have been modified by the data just stored.† For all but SS instructions, the ASC test is made during the execution phase whenever a store operation is performed. Because, in the case of SS instructions, a store operation is always implied, an ASC test has been incorporated in the SS I-Fetch microprogram.

†The refetch routine is initiated if the result of a comparison of the destination and IC addresses falls within a 16-byte safety margin: Destination address (in D) = IC address ± 16 bytes. Thus, instructions in Q may not necessarily be modified by the store operation.

For SS instructions, the ASC test must determine that instructions (currently in Q) were not obtained from a region defined by the upper and lower limits of the destination address for data. This test is made in two steps, as illustrated in Figure 3-3. The first step determines whether the lower limit of the destination address is above the instruction address in the IC. When the lower limit is above, the upper limit must also be above the IC, and the 'PSC' trigger is not set. This condition indicates that current instructions (in Q) cannot be affected by the subsequent store operations. However, if the lower limit of the destination address is found to be below the IC, the 'PSC' trigger is set and a further test must be made to establish that data will not be stored in the instruction path. The last step compares the upper limit of the destination address with the IC. If the IC is found to be above the upper limit, the 'PSC' trigger is reset. In such cases the subsequent store operations will not extend to the IC location. On the other hand, if the IC points below the upper limit, the 'PSC' trigger remains set, indicating that the subsequent store operations may affect the next instruction. Consequently, after execution of the instruction that caused the PSC condition, an exceptional condition microprogram is initiated to refetch the instructions in Q. The details of the refetch microprogram are described under "Program Store Compare Exceptional Condition".

I-Fetch Microprogram

- If request for new instructions has been generated at end op, I-Fetch routine requires 7 cycles; if not, I-Fetch requires 6 cycles.
- Setting of IC(21,22) at end op determines manner in which I-Fetch is performed.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of SS instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The first halfword of the SS instruction is transferred from R to E at the start of I-Fetch. The operand prefetch (initiated at end op) is then continued with the first halfword in E, the second halfword in Q, and the third halfword in Q [or in main storage if IC(21,22) = 10], as shown in Diagram 5-14, FEMDM.

The SS instructions require a 7- or 6-cycle I-Fetch. The actions initiated during the first I-Fetch cycle are governed by 1 of 8 possible ROS control words selected at end-op time. This selection depends on whether the B1 field of the instruction is zero and on the setting of IC(21,22). The setting of IC(21,22) establishes four distinct cases for the SS I-Fetch microprogram:

1. When IC(21,22) = 00, a 4-cycle storage request to refill Q is generated at end op. Because Q(48–63) contains

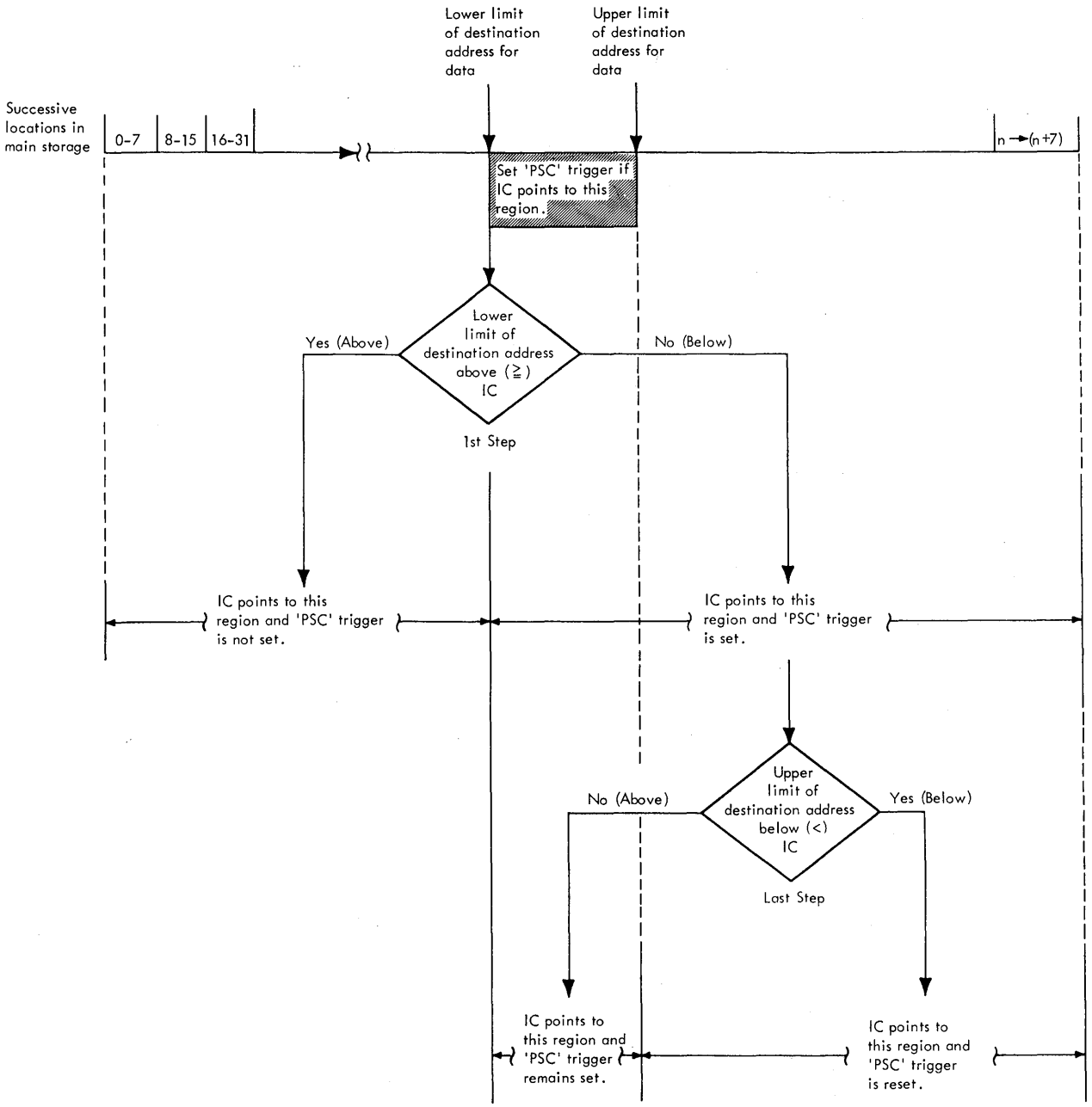


Figure 3-3. ASC Test for SS Instructions

the op-code halfword of the next instruction, the I-Fetch routine must transfer this halfword to R before the next doubleword arrives from main storage. Also, the I-Fetch routine must gate the new instructions to Q at the correct time.

2. When IC(21,22) = 01, a 4-cycle request to refill Q is generated. Because Q contains no new instructions, the I-Fetch routine must wait until Q is refilled and then transfer the first halfword of the next instruction from Q to R. New instructions must be transferred to Q at the correct time.
3. When IC(21,22) = 10, a 3-cycle request to refill Q is generated. Because the third halfword of the SS instruction is in main storage, processing of this halfword is delayed until Q is refilled. New instructions must be transferred to Q at the correct time, after which the next instruction may be gated to R.
4. When IC(21,22) = 11, a storage request is not generated because Q has been refilled as a result of the previous end op. Thus the I-Fetch routine is not concerned with transferring new instructions to Q.

During the first I-Fetch cycle of all SS I-Fetch routines, the lower limit of the destination address is calculated and placed into D, and IC(21,22) is advanced to indicate the first halfword of the next instruction. At the start of I-Fetch, T contains the base portion of the destination address (LS contents per B1). If B1 $\neq$ 0, the contents of T are added to the appropriate D1 field and the sum is transferred to D. If, however, B1 = 0, the contents of T are ignored and only the D1 field is routed via the parallel adder to D. After the first I-Fetch cycle, D contains the lower limit of the destination address.

The I-Fetch sequence after the first cycle differs for each setting of IC(21,22). The following paragraphs describe the I-Fetch control for each setting.

I-Fetch Control If at End Op IC(21,22) = 00

- STAT D is set to indicate B2 = 0.
- Branch per instruction is made to establish starting destination address.
- If 'execute in progress' trigger is reset, IC(20) is advanced by 1; if trigger is set, IC is not incremented.
- Instruction address is stored into LSWR.
- I-Fetch requires 7 cycles.

If at end op IC(21,22) = 00, Q contains the remaining two halfwords of the SS instruction and Q(48–63) contains the halfword of a new instruction. Q(48–63) is transferred to R by the second ROS word in the I-Fetch sequence. This word also accesses the LS register specified by B2 and initiates the ASC test. The LS register is accessed by gating the B2 field [Q(32–35)] to LAR and transferring the LS contents to S. (STAT D records the condition when B2 = 0 and is interrogated when the source address is computed.) The ASC test is initiated by subtracting the lower limit of the destination address (contained in D) from the IC. If the difference is equal to or greater than zero, the 'PSC' trigger is set to indicate that the destination address may overlap the instruction path.

A branch is made per the instruction op code to the third I-Fetch cycle. This branch establishes the manner in which the upper limit of the destination address must be obtained:

1. For decimal instructions, the L1 field [E(8–11)] is transferred to the parallel adder, where it is added to the contents of D.
2. For logical instructions, the LL field [E(8–15)] is added to the contents of D.

Also, depending on the instruction, either the upper or the lower limit of the destination address becomes the starting point from which operands are to be processed. For all decimal instructions, operand processing starts from the upper limit of the destination address and proceeds toward the lower limit. Conversely, for all logical instructions, operand processing starts from the lower limit and proceeds toward the upper limit. Thus, when a decimal instruction is decoded, the upper limit of the destination address is transferred to D and becomes the starting address from which the first operand will be requested. In the case of logical instructions, the original contents of D (lower limit of destination address) are not changed. Because the upper limit of the destination address is required to complete the ASC test, this address is temporarily stored into T.

The fourth ROS word in the I-Fetch sequence initiates calculation of the source address and gating of new instructions to Q, and requests the first operand from main storage (per the destination address in D). At the start of the fourth cycle, STAT D is tested to establish whether the B2 field of the instruction is zero. If B2 is not zero (STAT D not set), the contents of S (where LS contents per B2 have been placed) and the appropriate D2 field in Q are simultaneously gated to the parallel adder, and the sum is temporarily stored into B. If B2 = 0 (STAT D set), however, the contents of S are ignored and only the D2 field is placed into B. With the completion of the above actions, all halfwords in Q have been processed and Q is refilled with new instructions. The fourth ROS word also initiates a 4-cycle request per D for the destination operand. A ROS micro-order in the first GIS word gates the destination operand into the CE.

The fifth ROS word in the I-Fetch sequence completes the ASC test. The upper limit of the destination address (contained in T) is subtracted from the IC contents. If the difference is greater than zero, the 'PSC' trigger is reset. This condition indicates that the instruction address is above the highest main storage location into which data is to be stored. However, if the difference (IC minus T) is less than zero, the 'PSC' trigger remains set. The set state of the 'PSC' trigger initiates the program store compare exceptional condition after execution of the current SS instruction.

The sixth ROS word in the I-Fetch sequence usually increments the IC by 8 and then stores the updated address into the LSWR. Following each storage request from the IC, the contents of the IC must be updated by 8 to obtain the address of the next doubleword location from which subsequent instructions will be requested. An exception to this rule occurs if the SS instruction currently being processed is the subject of an Execute instruction. Because, upon execution of the subject instruction, the instructions previously contained in Q must be refetched from the main storage address specified in the IC, the contents of the IC are not updated. Thus, before storing the IC into the LSWR, a test must be made to determine whether an Execute instruction is in progress. This test is accomplished by examining the status of the 'execute in progress' trigger.

After the instruction address is transferred to the LSWR, the IC is ready to receive the address of the source operand. The seventh ROS word in the I-Fetch sequence transfers the lower limit of the source operand address (placed into B by the fourth ROS word) to the IC and to T. The 'RESET' micro-order resets the STC, ABC, STAT D, and the Edit-instruction controls, and sets the 'stop' trigger if operating at the instruction-step rate.

At the completion of the SS I-Fetch, a General Initialization Sequence (GIS) is performed. This sequence sets up various initial conditions that are required by logical or decimal SS instructions. The GIS is described in Sections 4 and 5 of this chapter.

I-Fetch Control If at End Op IC(21,22) = 01

If at end op IC(21,22) = 01, the balance of the SS instruction is contained in Q. The I-Fetch sequence for this case is similar to that performed if IC(21,22) = 00 at end op. However, the following differences exist.

Because no new instructions are contained in Q at the start of I-Fetch, loading of R with the halfword of the next instruction is delayed until Q is refilled. Because of the 4-cycle request generated at end op, Q is refilled on the fifth I-Fetch cycle; on the sixth I-Fetch cycle, Q(0-15) is transferred to R. (The 'MS→Q' and 'QXX→R' micro-orders are issued one cycle before the actual transfer.)

I-Fetch Control If at End Op IC(21,22) = 10

If at end op IC(21,22) = 10, the second halfword of the SS instruction is in Q(48-63) and the third halfword is in main storage. Processing of the third halfword of the SS instruction and loading of R with the halfword of the next instruction are delayed until Q is refilled. Because of the 3-cycle request generated at end op, Q is refilled on the fourth cycle of I-Fetch. (The 'MS→Q' micro-order is issued one cycle before the actual transfer.) Except for this difference in timing, the functions performed by the I-Fetch sequence are identical with those performed if IC(21,22) = 00 at end op.

If the fetching of the third halfword had resulted in an invalid address or a protection violation, the 'program interrupt' latch is found set during the fourth cycle. This suppresses both the remaining I-Fetch functions and the execution of the SS instruction. IC is incremented by 8 if the 'execute in progress' trigger is not set, and the CE clock is stopped one cycle to ensure that the SCI is not busy. The SS I-Fetch routine is then completed with a normal end op and the program interruption microprogram is forced.

I-Fetch Control If at End Op IC(21,22) = 11

If at end op IC(21,22) = 11, Q(0–31) contains the balance of the SS instruction. Because a request to refill Q is not generated, the I-Fetch sequence is not concerned with transferring new instructions to Q and incrementing the IC by 8. (Consequently, only 6 cycles are required to complete the I-Fetch routine.) Except for these differences, the functions performed by the routine are identical with those performed if IC(21,22) = 00 at end op.

DEVIATIONS FROM BASIC END OP AND I-FETCH

Detection of an interruption or exceptional condition at end-op modifies the basic end-op and I-Fetch sequences previously described. All interruptions and most exceptional conditions inhibit the following I-Fetch actions:

1. Transfer of the next instruction from R to E.
2. Incrementing of IC to select the next instruction.
3. Transfer of the next instruction from Q to R.

Two exceptional conditions do not modify I-Fetch in the above manner. These are (1) the exceptional condition that tests for invalid address, and (2) the Q-register refill exceptional condition. Because both of these conditions are detected during the first I-Fetch cycle (i.e., the cycle in which IC incrementing Q-to-R and R-to-E transfer takes place), these I-Fetch actions are not inhibited. However, detection of an invalid address test or a Q-refill exceptional condition overrides the 'E(02-07)→ROA' micro-order that may be in the first I-Fetch word and causes a branch to recovery microprogram.

To understand how an interruption or exceptional condition modifies the subsequent I-Fetch sequence, the reader must be familiar with the operation of the following I-Fetch controls: (1) the I-Fetch sequencers, and (2) the 'block I-Fetch' trigger. These controls are described in subsequent paragraphs.

I-Fetch Sequencers

- Three I-Fetch sequencers control Q refilling, IC(20) updating, instruction address checking, and ROS addressing of appropriate Q-refill exceptional condition microprogram.

The I-Fetch sequencers (Diagram 5-15, FEMDM) are a set of three trigger-latch circuits activated when a need to refill Q is detected by the 'NEOP', 'BEOP', or 'EEOP' word. The sequencer status indicates the number of cycles that have elapsed from the time of the request. The I-Fetch sequencers control:

1. IC(20) updating. The IC is updated when sequencer 2 is set and sequencer 1 is reset. This condition gates the contents of the IC to PAB(40-63) and, simultaneously, adds a 1 to PAA(60) to update IC(20). (This updating is equivalent to increasing the instruction address by 8.) The new instruction address is gated back to the IC.

2. Ingating of new instructions to Q. After a request to refill Q is initiated, the sequencers keep track of the CPU cycles to transfer new instructions to Q at the correct time. When sequencers 3 and 1 are both set, the new instructions are gated to Q on the next cycle. (Note that sequencer 1 is set by sequencer 2 and by the 'RASCR' micro-order contained in the first word of the Q-register refill exceptional condition microprogram if not a 2-cycle-early request.) If, when Q is refilled IC(21,22) = 00, the first halfword position in Q is automatically transferred to R. This transfer takes place because, at the time of request, the halfword of the next instruction was in main storage. Consequently, the transfer of the next halfword to R is delayed until this time.
3. Address checking. The 'I-Fetch request' trigger is set when sequencer 1 is set and sequencer 3 is reset. This condition initiates the invalid instruction address test described later when discussing that exceptional condition. Operand addresses are tested for specification violations from D, instead of the parallel adder, by the 'SPEC' micro-order during execution of non-indexed RS, SI, or RS instructions if a Q-register refill is still in progress (sequencer 2 set).
4. ROS addressing. The 'EXCEP' micro-order in the first I-Fetch word samples the sequencer status to establish whether a Q-register refill exceptional condition has priority. If Q-register refill priority is recognized, a new address is forced into ROSAR. This address is determined by the sequencer status and by the format of the upcoming instruction. The microprogram thus forced provides the time interval required to refill Q.

Block I-Fetch Trigger

- Set by detection of interruption or exceptional condition (except invalid instruction address test and Q-register refill).
- Resets I-Fetch sequencers and blocks most actions of first I-Fetch word; chiefly, incrementing of IC, R-to-E transfer, Q-to-R transfer, and main storage requests.

Upon entering the interruption-processing microprogram, the CE must contain the instruction that was responsible for the interruption. Except when a branch to a new address is anticipated, requests to refill Q are issued during end op and, sometimes, one or two cycles before end-op. On the other hand, the interruption-processing microprogram cannot start until the first I-Fetch cycle, at which time Q may be in the process of being refilled. For this reason, a means must be provided to inhibit the refilling of Q upon detection of exceptional conditions. This function is performed by the 'block I-Fetch' trigger. This trigger also inhibits those I-Fetch actions associated with the next instruction; namely, updating the IC, transferring R to E, and transferring Q to R.

The 'block I-Fetch' trigger (Diagram 5-16, FEMDM) is set if an interruption or exceptional condition is detected and the instruction responsible for this condition is contained in the CE.† This trigger inhibits the following I-Fetch actions:

1. Ingating of new instructions to Q. The output of the 'block I-Fetch' trigger resets the I-Fetch sequencers to prevent ingating of new instructions to Q. Note that this function is significant only during I-Fetch of 1- and 2-halfword nonbranch instructions. For branch and 3-halfword (SS) instructions, ingating to Q is performed by the ROS execution microprogram, and the sequencers are not activated; because upon detection of an exceptional condition this microprogram is not entered, ingating of new instructions to Q does not take place.
2. Updating of IC(21,22) or D(21,22). When a branch to an interruption-processing microprogram is performed, these bits must indicate the first halfword of the instruction that caused the interruption. The 'block I-Fetch' trigger inhibits these bits from being advanced to indicate the next instructions.
3. Updating of IC(20) or D(20). Because the trigger resets the I-Fetch sequencers, the address for the next instructions in main storage is not incremented.
4. Transferring R to E. The first halfword of the instruction that caused the interruption must be retained in E. The trigger prevents the halfword of the upcoming instruction from being transferred to E.
5. Transferring Q to R. Because the halfword of the next instruction remains in R, the halfword of the upcoming instruction must remain in Q. The trigger inhibits any 'QXXR' micro-order in the first I-Fetch word from effecting this transfer.
6. Main storage requests. Requests for new instructions because of a predecoded branch instruction are inhibited by blocking the decoding of the 'RESET' micro-order. Storage requests for operands are inhibited by blocking the decoding of the 'MS-REQ*D-3' micro-order.
7. Initiation of the invalid instruction address test.

The 'block I-Fetch' trigger is not set by detection of the invalid instruction address test or Q-register refill exceptional conditions. In the first case, because addressing is at fault, the next address must be computed and retained

†The only exception to this rule occurs when the Execute instruction is in progress. In this case, the interruption-processing microprogram cannot be entered until execution of the subject instruction (specified by the Execute instruction) is completed. Therefore, a request for the subject instruction is not blocked. Setting of the 'block I-Fetch' trigger is inhibited by the set states of the 'execute in progress' trigger and STAT G (set by the Execute instruction).

for subsequent evaluation. In the second case, the trigger is not set because this action would inhibit the purpose of the Q-register refill exceptional condition.

The 'block I-Fetch' trigger is reset by the '0→STAT D' micro-order issued by the first word in the recovery microprogram.

INTERRUPTIONS AND EXCEPTIONAL CONDITIONS

When an interruption occurs, the interruption code is recorded in the current PSW of the CE. This PSW is then stored in main storage as the "old" PSW, and the CE fetches a "new" PSW from main storage. Both the old and the new PSWs have fixed locations within the CE's PSA. (Recall that an interruption can be masked off by the interruption field in the current PSW.)

An exceptional condition operation does not change the current PSW, nor does the PSW contain mask bits for exceptional conditions. After the exceptional condition is processed, the instruction flow might be continued, stopped, repeated, or left, depending on the specific exceptional condition.

The subsequent paragraphs describe the specific hardware and microprogram sequences used for processing each interruption and exceptional condition. The discussions of the interruptions and exceptional conditions follow the order of priority in which these conditions are processed by the CE:

1. Timer exceptional condition:
   a. External Start or Stop
   b. Receive ATR
   c. Time clock step
2. CPU store in progress exceptional condition
3. Machine-check interruption
4. Program interruption
5. Supervisor call interruption
6. External interruption
7. I/O interruption
8. Manual control stop exceptional condition
9. Manual control wait exceptional condition
10. Manual control repeat exceptional condition
11. Program store compare exceptional condition
12. Invalid instruction address test exceptional condition
13. Q-register refill exceptional condition

Each interruption class (machine check, program, supervisor call, external, and I/O) requires individual treatment before handling by a common ROS routine. Accordingly, the discussion of these interruptions follows the same pattern; i.e., the individual handling for each interruption is described first, followed by a description of its common routine.

Timer Exceptional Condition

- Decrements timer in location 50 (hex) per power-line frequency.
- 'Time clock step' trigger is set when timer needs update.
- 'Time clock step' trigger can also be set by other conditions: external start, external stop, or receive ATR.
- Hardware controls cause machine reset if external start/stop; thus terminating timer update ROS routine.
- ROS branch on J27 micro-order (ATRSEL) handles receive ATR.

The timer exceptional condition occurs when the interval timer must be stepped (decremented). This condition is recognized by the 'time clock step' trigger, which is set by a positive swing of the line frequency.

The 'time clock step' trigger (Diagram 5-17, FEMDM) is also set by three other conditions: external start, external stop, and receive ATR. Any one of these conditions causes the CE to enter the 'time clock step' microprogram. However, only the true timer exceptional condition will cause timer update. The external start or external stop condition and the receive ATR condition take priority by overriding the timer update microprogram. The external start condition causes the CE to reset and perform a PSW restart routine. The external stop condition causes the CE to reset and go to the stopped state. The receive ATR condition causes the CE to gate in ATR data from another CE which is performing a Set Address Translator (SATR) instruction.

If none of these three conditions exist, the timer is updated as soon as the instruction in progress is finished. The CE generates main storage address 50 (hex), the location of the timer value. The timer value is fetched, stepped, and returned to location 50. If the timer value is stepped from a positive value to a negative value, a timer external interruption is processed next. Otherwise, processing continues with the next instruction.

The microprogram accessed by the 'time clock step' trigger is shown in Diagram 5-17, FEMDM. Each time the positive swing of the power-line ac voltage occurs, the 'sample pulse' trigger is set (via a 300-ns singleshot), provided the DISABLE INTERVAL TIMER switch is not activated. The 'sample pulse' trigger, in turn, sets the 'time clock step' trigger, provided the CE is not currently in an end-op cycle.

The need to inhibit setting of the 'time clock step' trigger at end-op is twofold: (1) the timer exceptional condition is asynchronous with respect to program

execution, and (2) it has the highest priority. Because priority is established at end-op, sampling the timer exceptional conditions at this time could result in a priority conflict with a pending interruption or exceptional condition. Thus, use of two triggers ensures that timer priority is present before entry into end-op; i.e., if the need to update the timer arises at end-op, this is recorded by the 'sample pulse' trigger, and the timer update microprogram is initiated on the next end-op cycle.

The 'time clock step' trigger may also be set by another CE executing a Direct Control External Start or Stop or a Set Address Translator (SATR) instruction. In any case, priority must be established for the timer exceptional condition. When end-op occurs, the trigger output is gated to alter the subsequent I-Fetch by inhibiting the loading of E from R. D is set to 50 (hex) preparatory to obtaining the timer value from main storage, and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects timer priority and forces address 014 (hex) into ROSAR. This is done without regard for the particular condition that sets the 'time clock step' trigger. Forcing address 014 in ROSAR causes the timer update microprogram to be entered.

The 'block I-Fetch' trigger is reset by the 0 → Stat D micro-order in ROS word 036 of the timer update microprogram. A J27 micro-order (ATRSEL) in this same ROS word performs two functions. It is ANDed with external start or stop, and it represents a ROS branch condition if a receive ATR condition is present.

If an external start or stop condition exists when the ATRSEL micro-order is encountered, a machine reset is hardware-generated as a result of the ANDing mentioned previously. The reset forces ROS to the stop loop and, therefore, terminates the timer update microprogram. If the reset is caused by an external start condition, a branch from the stop loop occurs, and a PSW Restart (Diagram 5-601, FEMDM) is performed; otherwise, the CE remains in the stop loop.

If no external start or stop condition exists, the ATRSEL micro-order causes the CE to test for a receive ATR condition. If the receive ATR condition is present, the receive ATR ROS routine is entered (Diagram 5-809, FEMDM); otherwise, the CE continues the timer update ROS routine.

The microprogram issues a three-cycle storage request per D to fetch the timer value. While the fetch is in progress, any protection checks from storage are ignored. After the fetch, the 32-bit timer value is loaded into A and, then, decremented by 5. The updated timer value is placed into S and T and, then, stored per the D-address (location 50, hex). Before storage, however, the timer value is sampled to see if it has been decremented to less than 0. If this condition exists, the 'time clock at limit' latch is set to request an external interruption on the following end op.

CPU Store in Progress Exceptional Condition

The CPU† store in progress exceptional condition accommodates the unique situation in which a store operation is in progress at end-op. Since interruptions and exceptional conditions are detected during this end-op cycle, the CE would be unable to record a "late" storage protection violation if one occurred. To prevent this situation, a special circuit is provided to test for a store-in-progress condition at end-op (Diagram 5-18, FEMDM). If an exceptional condition to I-Fetch or a Load PSW instruction has been detected while a store operation is in progress, this circuit forces a microprogram that provides a two-cycle delay; this allows recording of a possible protection check and establishment of the correct priority for the subsequent program interruption. If a protection check occurs, a program interruption is processed next. Otherwise, processing continues with the next instruction.

Machine Check Interruption

- Follows log-out microprogram.
- Sets interruption code to identify cause, resets STAT II, and enters common interruption routine.

A machine check interruption is initiated by a logout operation, by an RDD (read direct) timeout or by an IOCE machine check request. The machine check interruption detection scheme is shown in Diagram 5-19, FEMDM.

When the logout operation (initiated by the Diagnose instruction or a machine check error coupled with the machine check mask-bit being on) is about to be concluded, the '1→MCH-CK-TRP' micro-order causes the 'machine check interrupt' trigger to be set. The RDD Timeout or IOCE Machine Check request do not cause a logout in the CE but set 'machine check' trigger directly. Once set, the trigger blocks any new machine checks from initiating another logout operation. This trigger also establishes machine check priority, and the operation then waits for the logout to finish (signified by an end op) before continuing. At end-op time, the set state of the 'machine check interrupt' trigger forces D to 30 (hex) and, if an

†CPU stands for Central Processing Unit, a term carried over from other computing systems. It is retained in the ALDs and refers, generally, to the CE.

Execute instruction is not being concluded, sets the 'block I-fetch' trigger, inhibiting most of the I-fetch actions.

The 'EXCEP' micro-order in the first I-fetch cycle detects machine check priority and forces address 00C (hex) into ROSAR. The ROSAR address causes a branch to an interruption microprogram that stores the old PSW per the D-address. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). The operation then proceeds as directed by the new PSW.

A flowchart of the hardware operations just explained and of the beginning of the machine check microprogram is shown in Diagram 5-19. The microprogram starts by gating the interruption code to S(16 -31) to begin forming the old PSW in ST. This is done by setting S(16 -31) to all 0's if the machine check was caused by the CE. If it was caused by an RDD Timeout, bit 29 is set and if the IOCE machine check request caused the interrupt, bits 30 31 encode the IOCE identity. The 'block I-fetch' trigger and STAT H are reset. (STAT H is reset to modify the subsequent common interruption routine for specific machine check actions.) At this point, the machine check interruption microprogram enters the common interruption routine.

Program Interruption

- Initiated by 'program interrupt' latch.
- Value in interrupt code triggers is transferred to S(16 31) by 'priority 1' latch.
- Sets STAT H, enters common interruption routine.

A program interruption results from improper conditions arising during the processing of data or instructions. Generally, these improper conditions can be described as errors in programming. When any of these conditions are detected, they cause a value to be placed into the eight 'interrupt code' triggers. The value inserted reflects the condition responsible for the interruption. The conditions that cause a program interruption and their corresponding Interrupt Code trigger settings are shown in FEMDM Diagrams 5-20, 5-21, and 5-22.

Once any Interrupt Code trigger has been set, the 'program interrupt' latch shown in Diagram 5-22 is set. (The 'INTRP X-branch' micro-order samples this latch to modify the execution sequence of instructions and the I-Fetch of SS format instructions. Once this latch is set, further program violations are lost with the exception of a "late" protection check.) This latch establishes priority for the program interruption by blocking the priority circuits of lower-priority interruptions and exceptional conditions.

At end op, the output of the 'program interrupt' latch is gated to force D to 28 (hex), to set the 'priority 1' latch, and, if an Execute instruction is not in progress to set the 'block I-Fetch' trigger. The 'EXCEP' micro-order in the first I-Fetch cycle detects program interruption priority and forces address 00A (hex) into ROSAR. This ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 1' latch causes the values in the Interrupt Code triggers (program-interruption code) to be gated to S as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CE operation then proceeds as dictated by the new PSW.

Diagram 5-22 is a flowchart of the hardware operations just described and of the beginning of the program interruption microprogram. The microprogram starts by gating the interruption code from PSW(16–31) to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. (STAT H modifies the subsequent common interruption routine for specific program interruption actions.) At this point, the program interruption microprogram enters the common interruption routine.

Supervisor Call Interruption

- Initiated by 'supervisor call' trigger, which is set by preceding Supervisor Call instruction.
- E(8–15), which contains interruption code, is transferred to S(24–31).
- Sets STAT H, enters common interruption routine.

The supervisor call interruption results from execution of the Supervisor Call instruction. Its basic purpose is to initiate a branch to the supervisor program. When the priority of the interruption is established, an address is forced into ROSAR and into D. The address in ROSAR causes the operation to branch to a microprogram which stores the old PSW in the address forced into D and fetches a new PSW from the address in D + 40 (hex). This new PSW places the CE into the Supervisor state.

Diagram 5-23, FEMDM, shows how the '1→ INTREQ-TGR' micro-order tests for a Supervisor Call instruction and sets the 'supervisor call' trigger. This trigger is reset if the 'interrupt code 4' trigger is set. Because all program interruptions would have been handled before executing the Supervisor Call instruction, the 'interrupt code 4' trigger can be set now only by a "late" protection check. Therefore, performance of the supervisor call interruption is suppressed and a program interruption occurs in its place. If the 'interrupt code 4' trigger is not set, then the 'supervisor call' trigger is not reset.

The 'supervisor call' trigger gates E(8–15) to S(24- 31) to begin assembling the old PSW. (The 'supervisor call' trigger also sets the 'priority 1' latch; because the supervisor

call and program interruptions cannot be pending at the same time, no conflict results from both setting an interruption priority code of 01.)

Diagram 5-23 is a flowchart of the hardware operations just described and of the beginning of the supervisor call interruption microprogram. The microprogram starts by gating the interruption code from PSW(16 31) to S(16- 31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. At this point, the supervisor call interruption microprogram enters the common interruption microprogram routine.

External Interruption

- Remains pending if external mask bit, PSW(7), is not set, with the exception of 'force interrupt', which bypasses the mask bit, PSW(7).

- Initiated by one of the following:
  1. Setting of 'time clock at limit' latch.
  2. Depression of INTERRUPT pushbutton.
  3. Setting a bit in DAR, for which there is a corresponding bit on in the DAR mask.
  4. Setting a bit in the PIR, for which there is a corresponding bit on in the CCR.
  5. Receiving one of six RDD or WDD signals from another CE.

- 'Time clock at limit', 'console' signal, 'CE RDD' and 'WDD' signals, 'DAR Interrupt' and 'PIR interrupt' triggers are transferred to S(20–31).

- Sets STAT H, enters common interruption routine.

An external interruption is caused by one of the following if the external bit of the PSW system mask is a 1:

1. The set state of the 'time clock at limit' latch (refer to "Timer Exceptional Condition").
2. The depression of the INTERRUPT pushbutton on the system control panel.
3. The recognition of any signal on the 'external signal in' bus of the Direct Control feature. The external interruption circuits and a flowchart of the initiation of the external interruption microprogram are shown in Diagram 5-24, FEMDM.

Once priority for the external interruption is established during end op, D is forced to 28 (hex), the 'priority 2' trigger is set, and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects external interruption priority and forces address 006 (hex) into ROSAR. The ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 2' trigger causes the contents of the eight signal triggers (the interruption code) to be gated to S(20 31) as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CE operation then proceeds as dictated by the new PSW.

The external interruption microprogram starts by gating status of the interrupt triggers to S(20 31). S(16 19) is reset to zero, and correct (odd parity is assigned to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. (STAT H is set so that the common interruption routine skips micro-orders pertaining to the machine check interruption.) At this point the microprogram enters the common interruption routine.

I/O Interruption

- Remains pending at IOCE if associated channel mask bit, in PSW(0 -6, 16- 19), is not set.

- IOCE 1 has highest interruption priority, followed in order by IOCE 2 and 3.

- Three-bit channel address and eight-bit unit address are transferred to S(21–31).

- Sets STAT H; sends PSA address to IOCE, causing interruption.

An I/O interruption results from the reception of a simplexed 'interruption request' from an IOCE. When operating in 9020 mode, the CE makes its system mask available to the IOCE at all times. The IOCE determines whether to initiate an interrupt request by examining this mask.

Channel priority is determined by hardware within the IOCE. In each IOCE, the multiplexor channel (channel 0, 4, 8) is assigned the highest priority, followed in order by selector channels (1, 5, 9), (2, 6, A), (3, 7) for IOCE's 1 3.

The priority of IOCE interruption requests is determined by hardware within the CE, with IOCE 1 having the highest priority, followed by IOCE 2, and with IOCE 3 having the lowest priority. The I/O interruption circuits and a flowchart of the initiation of the I/O interruption microprogram are shown in Diagram 5-25, FEMDM. At the start of end op, the highest-priority trigger that is set resets all the others.

Once priority for an I/O interruption is established during end-op, D is forced to 38 (hex), the 'priority 1' and 'priority 2' triggers are set, and, if an Execute instruction is not in progress, the 'block I-fetch' trigger is set. The 'EXCEP' micro-order in the first I-fetch cycle detects I/O interruption priority and forces address 00E (hex) into ROSAR. This ROSAR address causes a branch to an interruption microprogram to store the old PSW per D.

For I/O interrupts, the CE and IOCE share the responsibility for storing the I/O old PSW, in the proper PSA area. The CE stores bits 0–15 and bits 32–63. It sends the proper physical address of the I/O old PSW location to the IOCE that initiated the request. The IOCE stores the interruption code (bits 20–31) containing the channel and device address in bits 20–31 of the I/O old PSW location. At the same time, the IOCE, in effect, propagates the extended system mask (bits 16–19 of the current PSW in the CE, which is available to all IOCE's via distributed simplex lines) into bits 16–19 of the old I/O PSW area, thus completing the formation of the 64-bit I/O old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CE operation then proceeds as dictated by the new PSW.

The I/O interruption microprogram starts by taking the contents of physical PSBAR and logical PSBAR and combining them to provide the address to be shipped to the IOCE (via the External register) to point to the PSA area where the IOCE is to store the interruption code. Then, the CE sends 'permit interrupt' to the IOCE and waits in a timing loop for a 'response' signal from the IOCE. After the receipt of 'response', the CE stores the remainder of the I/O old PSW (bytes 0, 1, and 4–7). STAT H is set so that the common interruption routine, which follows next, skips micro-orders pertaining to the machine check interruption.

Common Interruption Routine

- Program status is assembled in ST and is then stored into old PSW location per interruption cause.
- System is reset if STAT H is reset (machine check interruption).
- Applicable new PSW is fetched per D.
- Processing resumes after new instructions have been fetched and placed into Q.

The common interruption routine (Diagram 5-26, FEMDM) stores the old PSW into main storage and loads a new PSW into the CE. This routine is entered by all interruption microprograms, except I/O interruptions.

The IC is reduced by 8 or 16 to reflect the doubleword address of the instruction that caused the interruption.

If the '$\triangle$IOERR' branch condition is met, indicating an 'IOCE MCH CK', a branch is taken to the I/O interruption microprogram. This address is placed into T(40–63) as part of the old PSW. Next, the contents of the PSW register are gated to S(0–15) and T(32–39). This action completes the old PSW transfer to ST. The routine inhibits storage protection, sets marks 0–7, and initiates a four-cycle storage request to store the old PSW per the D address. An interruption reset clears the CE of the condition that initiated the access to the interruption microprogram. The 'invalid branch' trigger, the 'invalid instruction address' trigger, and STAT G are reset.

At this point, the common interruption routine checks STAT H to see what class of interruption initiated the operation. If STAT H is reset, the operation is due to a CE machine check interruption, and the CE is placed in the scan mode. Three no-op cycles are taken to allow the CE and main storage to become quiescent. Then a 'system reset' signal clears all control triggers. The 'scan mode' trigger is reset, and the routine prepares to load the new PSW.

STAT H is set if the initiating interruption is other than a machine check. In this case, the CE is not placed in the scan mode and the system is not reset.

To generate the address for the new PSW, 10 (hex) is placed into B, setting B(59) to 1. Next, the value in B is shifted left twice and gated to PAB as an effective value of 40 (hex). Simultaneously, the old PSW address is gated from D to PAA. The sum, the address of the new PSW, is gated from PAL to D. Storage protection is then inhibited, and a 3-cycle fetch per D is initiated.

The interruption microprogram has now finished the common interruption routine and enters the Load PSW microprogram (Diagram 5-601, FEMDM). When received from main storage, the new PSW is loaded into ST. Because the new PSW will require fetching of instructions from a new storage location, the 'I-Fetch invalid address' trigger is set to enable recording of any invalid address that may result on the subsequent fetch. Portions of the new PSW are loaded into the PSW register and into the IC and D. A 3-cycle storage request per the IC is initiated, and the IC is incremented by 8. At this point, the program shifts to a common branch microprogram. The instruction address in D is incremented by 8 and transferred to the IC in anticipation of a branch instruction. When the first doubleword arrives from main storage it is loaded into Q, and the op-code halfword of the first instruction is transferred to R. D(21,22) is sampled to see if Q needs refilling; if so, a second request per the IC is initiated. If Q does not need refilling, the program generates an end op.

Stop, Wait, and Repeat Exceptional Conditions

The stop exceptional condition is caused by (1) depressing STOP, (2) detecting an address-compare condition when ADDRESS COMPARE STOP is in the STOP position, and (3) operating at the instruction step rate. The CE enters a "stop loop" during which no instructions are processed and all interruptions are kept pending. Certain pushbuttons and external conditions are sampled in the stopped state, which, if active, can cause the CE to exit from the stop loop. Pushbutton sampled are: STORE, DISPLAY, SET IC,

START, ROS TRANSFER, REG SET, LOAD, and PSW RESTART. The external conditions tested are IPL (system or subsystem), external start, and ATR select.

The wait exceptional condition is caused by the wait mask bit, PSW(14), being set to a 1. CE clock signals are inhibited. Processing continues when an external or I/O interruption, an external start, or an IPL operation is initiated.

The repeat exceptional condition arises during the repeat instruction operation (a maintenance aid). The REPEAT INSN (instruction) switch on the CE control panel must be in the SINGLE position.

The scheme for detecting a stop, wait, or repeat exceptional condition is shown in Diagram 5-27, FEMDM.

When any one of these exceptional conditions has priority during end-op and an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. During the next cycle, the first I-Fetch cycle, the 'EXCEP' micro-order forces an address into ROSAR: 026, 02A, and 028 (hex) for the stop, wait, and repeat exceptional condition, respectively. This address causes a functional branch to a loop microprogram. Because each of these exceptional conditions may be caused by manual intervention, their microprograms are discussed in Chapter 4, Section 1.

Program Store Compare Exceptional Condition

- Instruction refetch is performed when 'PSC' trigger is set.
- Refetch routine decrements instruction address by 8 or 16 and issues request to refill Q.

A program store compare exceptional condition results if the next instruction to be processed must again be fetched into the Q-, R-, and E-registers. This need occurs after processing of the Execute, Diagnose or Load PSBAR instructions and after some store operations. Although the instruction to be executed next is held in the Q-register, this instruction has been modified in its main storage location. Therefore, to have the correct version of the instruction in the Q-register, the instruction must be refetched.

An instruction refetch is initiated by the 'PSC' trigger. The scheme for detecting a program store compare exceptional condition and the instruction refetch microprogram flowchart are shown in Diagram 5-28, FEMDM.

A need to refetch instructions is treated as an exceptional condition by the CE. When this condition is detected, the 'block I-Fetch' trigger changes the normal I-Fetch routine, and a branch to an instruction refetch microprogram is performed by the first I-Fetch word.

The first ROS word in the refetch microprogram resets the 'block I-Fetch' trigger so that normal I-Fetch can be resumed after Q is refilled. This word also establishes whether the address currently specified in the IC is one or two doublewords ahead of the current instruction. The address in the IC is always at least one doubleword ahead of the address for the instructions in Q. If Q was not refilled before the refetch routine, the IC is one doubleword (8 bytes) ahead of the current instruction; if Q was just refilled, IC is two doublewords or 16 bytes ahead.

IC(21,22) indicates whether Q was refilled before the refetch routine. If IC(21,22) is not set to 11, a request to refill Q (if generated) was blocked by the exceptional condition in progress (i.e., the need for instruction refetch) and the IC is 8 bytes ahead of the current instruction. If, however, IC(21,22) = 11, the need for an instruction refetch occurred after Q was refilled; IC(20) has been incremented, and the IC is 16 bytes ahead. Accordingly, the second ROS word in the refetch microprogram subtracts 8 or 16 from the IC and issues a 3-cycle request per the decremented address. This word also resets the 'PSC' and 'execute in progress' triggers and then causes the Load PSW microprogram to be entered (as shown in Diagram 5-601, FEMDM). Entry corresponds to a point after the new PSW has been loaded but before the successful branch routine. IC is incremented by 8, the next instruction is transferred to R, and Q is refilled if necessary, before completing the program store compare exceptional condition microprogram with an end op.

Invalid Instruction Address Test Exceptional Condition

- Determines interrupt code triggers to be set if program check was detected while addressing instruction.

The previously discussed exceptional conditions and interruptions result from unusual conditions occurring during execution of an instruction. In an invalid instruction address test exceptional condition, however, it is the address of the next instruction that is invalid, protected, or incorrectly specified. Thus, an interruption cannot occur until the CE attempts to execute that instruction.

An invalid instruction address is one that fails to meet one or more of the following three requirements:

1. Because instructions are specified on a 2-byte basis, the least significant bit of the instruction address must always be a zero. Failure to meet this requirement results in a specification program interruption.
2. The instruction address cannot exceed the storage capacity used with a given installation. In addition, the storage unit containing the instruction must be available to the CE. An attempt by the CE to execute an instruction from an unavailable or nonexistent location

results in an addressing program interruption (Diagram 5-29).

3. The instruction address cannot specify an area in main storage that is fetch-protected. An attempt by the CE to execute instructions from a fetch-protected location results in a protection program interruption.

If any of these three requirements is not met, the CE hardware forces a new address into ROSAR. The microprogram accessed by this address sets the appropriate program interruption code (specification, addressing, or protection) into the CE. This microprogram is then followed by the program interruption microprogram previously described. The following paragraphs describe the methods used to detect each violation and to set the appropriate interrupt code triggers.

Specification Detection

Not all storage requests for instructions result in Q being refilled. For example, end-op requests are ignored if the 'block I-Fetch' trigger is set. Also, branch requests made during I-Fetch are ignored if the conditions for a successful branch are not met during the following execution (non-branch on condition instructions only). For this reason, the least significant address bit of the instruction, IC(23) [or D(23) if preceded by the 'BEOP' micro-order], is detected at the start of I-Fetch. However, because the specification interruption code is not yet set, the program interruption microprogram cannot be immediately entered. Instead, the invalid instruction address test exceptional condition microprogram is entered after processing all interruptions and higher-priority exceptional conditions. If, however, during this forced microprogram, an invalid or fetch-protected address is requested, the specification interruption code is not set because in either case, the address is outside of "fetchable" storage.

During execution of the Display instructions, if the new page address is not on a doubleword boundary, a microprogram branch is forced to address 007 (hex). It is then restored from the LSWR and a specification error is set. At end-op, the excep branch is made to the program interrupt microprogram.

Invalid Address Detection

- 'I-Fetch request' trigger prevents setting of addressing interruption code while refilling Q.
- 'I-Fetch invalid address' trigger indicates IC request is invalid.
- 'Branch invalid address' trigger indicates branch address of successful branch instruction is invalid.

Following a request to refill Q, the IC is incremented by 8 to obtain the instruction address for the next request. The scheme of incrementing the IC ahead of time allows greater speed in requesting instructions from main storage. However, with the IC one doubleword ahead of the instructions in Q, a unique case occurs if the instructions in Q are obtained from the last available location in main storage. In this case, the incremented IC specifies an invalid address; i.e., an address that is in excess of the main storage capacity. Because the Q-register refill routine is initiated before the CE runs out of instructions, a request per the IC refills Q with instructions from an invalid address.† Even though Q contains invalid instructions, an addressing program interruption must not occur until the CE attempts to process these instructions. This condition arises because the last valid instruction being processed by the CE may result in a successful branch to a valid storage location.

A similar situation may occur following an unsuccessful branch instruction that specifies a branch to the last available main storage location. Excluding the Branch on Condition instructions the CE assumes that the branch instruction is successful and, accordingly, issues a request per D. Following the request, D is updated by 8 and specifies an invalid address. In this case, an addressing program interruption must not occur because, upon establishing that the branch is unsuccessful, the CE resumes normal addressing per the IC. For branch instructions, an addressing interruption must occur only when the address specified by a successful branch is above the available main storage capacity. This situation may also exist after any load-PSW operation or after the program store compare exceptional condition.

An invalid-address test is performed each time the CE issues a request to refill Q. Because a request for invalid instructions will not necessarily cause an interruption, setting of the interrupt code triggers must be blocked while Q is being refilled.†† The scheme used for detecting a "true" invalid instruction address error is shown in Figure 3-4.

The 'I-Fetch request' trigger prevents the invalid-address condition from causing an interruption while Q is being refilled. This trigger is set when a need to refill Q is

†The instruction address is considered invalid by the SCI upon detection of a carry from the most significant bit position in the IC. This bit position is defined by the size of the main storage in the particular installation.

††An exception to this rule occurs if the last one or two halfwords of an SS instruction are requested from an invalid address while the first halfword is contained in a valid storage location. In this case, the entire SS instruction is considered to have an invalid address, and, because the CE has attempted to process the instruction, the interrupt code triggers are set as soon as the request is generated.

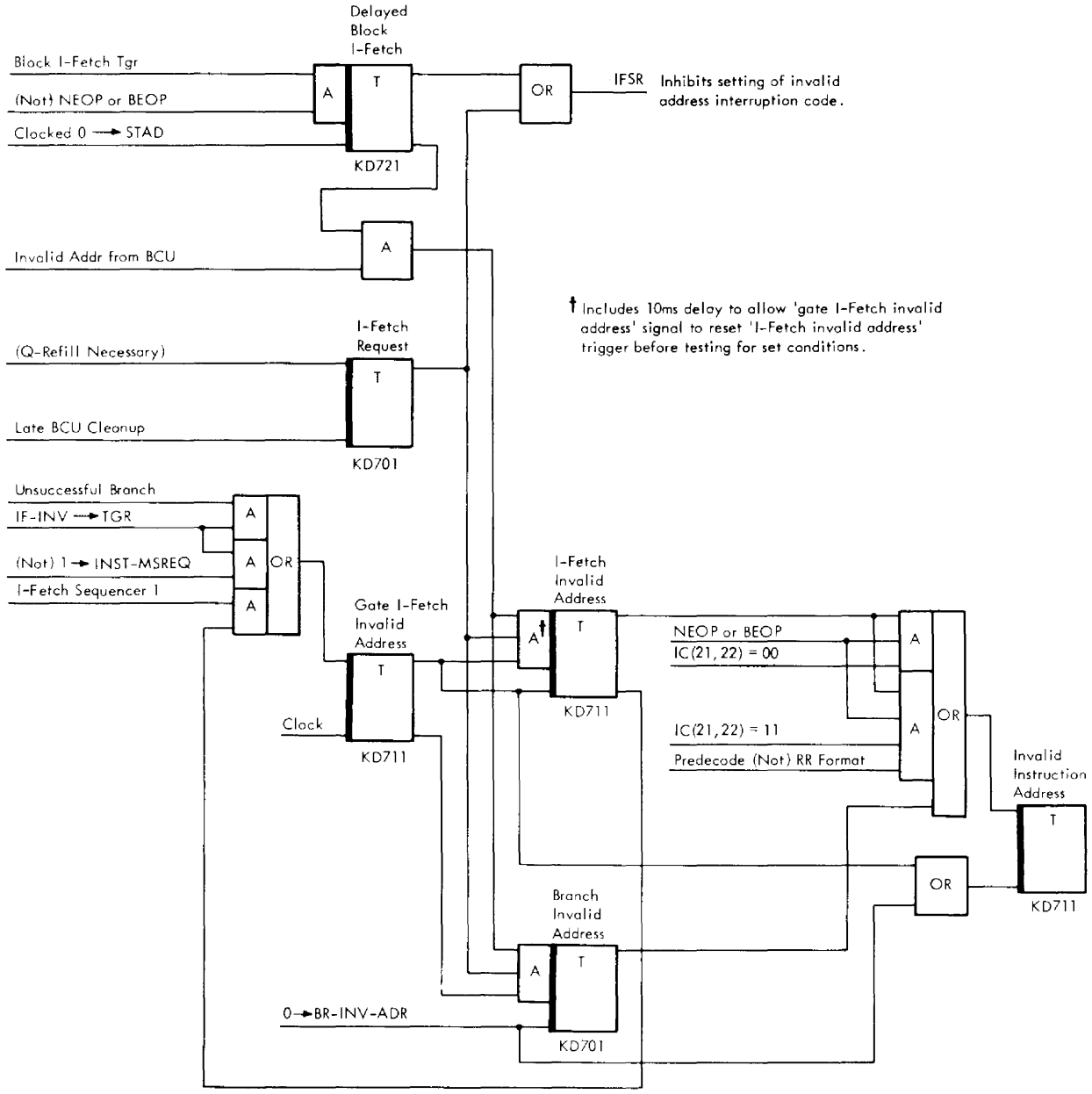


Figure 3-4. Detection of Invalid Instruction Address

detected; depending on the current instruction status in the CE, the trigger is set as follows:

1. For non-branch 1- and 2-halfword instructions, the trigger is set by the I-Fetch sequencers.
2. For branch instructions, the trigger is set by the '1→INST-MSREQ' micro-order given at the start of the branch execution. Note that if an unsuccessful Branch on Condition instruction occurs and IC(21,22) = 00, the 'I-Fetch request' trigger is not set because Q will not be refilled.
3. For SS instructions, the trigger is set if IC(21), or D(21) for the 'BEOP' micro-order, is equal to 0 during end op. This condition indicates that the complete SS instruction is already in Q and succeeding instructions are being requested.

The output of the 'I-Fetch request' trigger prevents the interrupt code triggers from being set by the 'invalid address' signal from the SCI. In addition, the output of the 'I-Fetch request' trigger serves as one of the conditioning

inputs for the 'I-Fetch invalid address' and 'branch invalid address' triggers. One of these triggers is set whenever the SCI indicates that the address of the storage request exceeds the main storage capacity. The 'invalid address' signal sets the 'I-Fetch invalid address' trigger if the invalid address is due to the CE fetching ahead. Conversely, this signal sets the 'branch invalid address' trigger when the invalid address is the result of a successful branch instruction.

The 'gate I-Fetch invalid address' trigger dictates whether the 'I-Fetch invalid address' or 'branch invalid address' trigger is to be set. When set, this trigger conditions the 'I-Fetch invalid address' trigger; when reset, the 'branch invalid address' trigger. Depending on the current instruction status in the CE, the 'gate I-fetch invalid address' trigger is set as follows:

1. For non-branch 1- and 2-halfword instructions, the trigger is set by the I-Fetch sequencers.
2. For SS instructions, the trigger is set by the 'IF-INV→TGR' micro-order given at the start of the SS I-Fetch routine; i.e., the presence of the 'IF-INV→TGR' micro-order and the absence of the '1→INST-MRSEQ' micro-order when the complete SS instruction is in Q.
3. For unsuccessful branch instructions (except Branch on Condition), the trigger is set by the presence of the 'IF-INV→TGR' micro-order and the absence of the '1→INST-MSREQ' micro-order; for an unsuccessful Branch on Condition instruction, the trigger is set by the simultaneous presence of the '1→INST-MSREQ' and 'IF-INV→TGR' micro-orders at the start of execution. (The various branching conditions that may arise are described in Section 6 of this Chapter.)

When the 'gate I-Fetch invalid address' trigger is set, the 'invalid address' signal is allowed to set the 'I-Fetch invalid address' trigger, indicating that Q has been refilled with instructions from an invalid address. However, because R may still contain a valid RR instruction, further testing is required to establish that a true interruption condition exists. The setting of IC(21,22) during end op indicates whether a valid or an invalid instruction is contained in R. If IC(21,22) = 11 and an RR instruction is predecoded, R contains a valid instruction. When IC(21,22) = 11 but the instruction is not of the RR format, the balance of the instruction has been obtained from an invalid location and the 'invalid instruction address' trigger is set. If IC(21,22) = 00, the 'invalid instruction address' trigger is set regardless of the instruction format; this condition indicates that R contains the first halfword of an invalid instruction.

When the 'gate I-Fetch invalid address' trigger is not set, the 'invalid address' signal sets the 'branch invalid address' trigger. Because, in this case, the invalid address is the result of a successful branch instruction, the 'invalid instruction address' trigger is set without further testing being necessary.

Fetch Protection Detection

- 'Delayed I-Fetch storage request' trigger prevents setting of protection interruption code while Q is being refilled.
- 'Delayed I-Fetch protect gate' trigger is set if request is due to normal sequencing.
- 'Protected branch address' trigger is set if branch is made to protected location.

The CE cannot execute instructions from a fetch-protected area in main storage. Because the IC is always one doubleword ahead of the instructions in Q, a unique situation occurs if the instructions in Q are obtained from a main storage location adjacent to a protected area. In this case, the incremented IC specifies a protected address and, because the Q-register refill routine is initiated before the CE runs out of instructions, the request per the IC refills Q with instructions from a protected address. A protection interruption, however, does not occur until the CE attempts to execute the protected instructions. This condition arises because the last valid instruction being processed by the CE may result in a successful branch to a valid storage location.

A protection test is performed each time the CE issues a request to refill Q. Because a request for protected instructions will not necessarily cause an interruption, the setting of the 'protection check (to CE)' and 'instruction length not available' triggers is blocked while Q is being refilled. The scheme for detecting a "true" protection violation, shown in simplified form in Figure 3-5, is closely related to the invalid addressing detection scheme previously described. The major difference between the invalid-addressing and fetch-protection schemes is in timing: the 'invalid address' signal arrives at the CE 1 cycle after the request, while the 'protection check' signal arrives 2 cycles after the request. For this reason a separate circuit is used for detecting a protection violation.

The 'delayed I-Fetch storage request' trigger prevents the 'protection check' signal from causing an interruption while Q is being refilled. This trigger is set one cycle after the 'I-Fetch request' trigger is set by a request to refill Q. The output of the 'delayed I-Fetch storage request' trigger serves as one of the conditioning inputs for the 'delayed I-Fetch protect gate' trigger. This trigger is set if the current storage request is due to a nonbranch or unsuccessful branch request and is set by the same conditions that set the 'gate I-Fetch invalid address' trigger. When the 'delayed I-Fetch storage request' trigger and the 'delayed I-Fetch protect gate' trigger are both set, the 'protection check' signal sets the 'I-Fetch invalid address' trigger. The action at this point is identical to that described for detection of invalid addressing; i.e., a test is made to establish if the CE has attempted to execute instructions from a protected

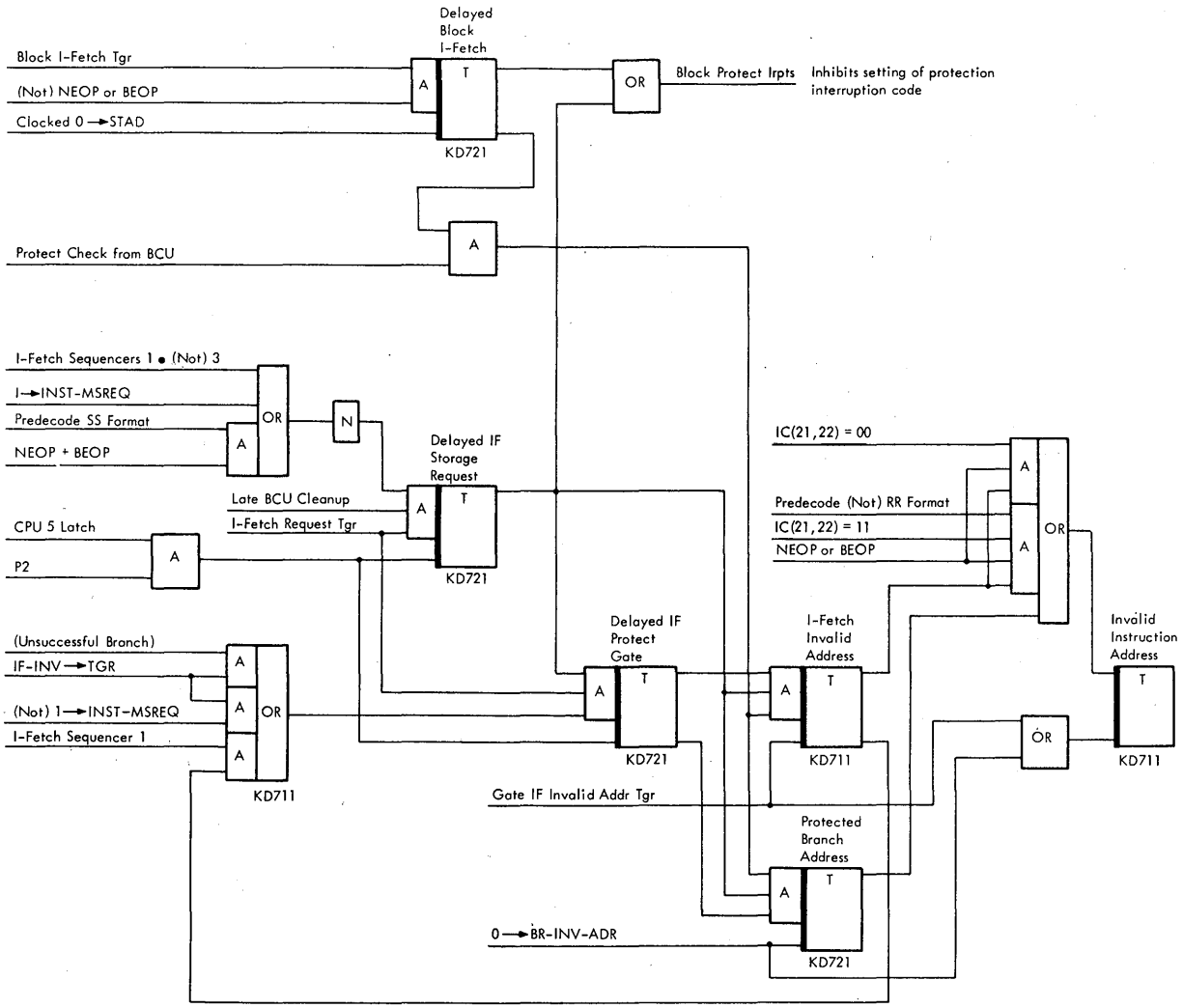


Figure 3-5. Detection of Fetch-Protected Instruction Address

area, and the 'invalid instruction address' trigger is set if a protect violation has occurred.

If the request to refill Q is a result of a branch instruction (as indicated by the reset state of the 'gate I-Fetch invalid address' trigger), the 'delayed I-Fetch protect gate' trigger is not set. In this case, the 'protection check' signal sets the 'protected branch address' trigger, which in turn sets the 'invalid instruction address' trigger.

Invalid Instruction Address Microprogram

- Issues second request for instruction.
- Interrupt code triggers are set per highest-priority error detected: (1) addressing = 101, (2) protection = 100, (3) specification = 110.

The detection of a specification, addressing, or protection exception and the associated microprogram are shown in Diagram 5-29, FEMDM. Because all three exceptions access the same microprogram, the microprogram must re-establish the nature of the exception to set the proper interruption code. To establish which exception is currently in effect, the microprogram issues a second request for instructions. This time, however, the setting of the interrupt code triggers is not blocked; i.e., the appropriate interruption code is set immediately upon detection of a specification, addressing, or protection exception.

Before generating a second request, the IC must be decremented to the address that caused the exception. The status of IC(21,22) indicates whether the current IC count is one or two doublewords ahead of the required address: if IC(21,22) is not set to 00, the IC is one doubleword ahead; if IC(21,22) = 00, the IC is two doublewords ahead. (The other recovery microprograms test IC(21,22) for a setting of 11; because the invalid instruction address test exceptional condition does not set the 'block I-Fetch' trigger, IC(21,22) is updated and tested for a setting of 00.)

Accordingly, the microprogram subtracts 8 or 16, decimal, from the IC, and loads the decremented address into D. The STC is then set to zero and a 3-cycle request is issued per D. After the request, the STC is incremented once during each subsequent cycle to provide the required wait interval between the request and what would be the cycle for transferring the instructions to Q. The appropriate interrupt code trigger(s) is then set upon receipt of a specification, addressing, or protection exception. Because the low-order bits of the addressing interruption code equal 101, there is no need to block the protection interruption code (100) from also being set if it is received. However, if either of these conditions is set, the setting of the specification interruption code, 110, is blocked. The specification interruption code need only be set if IC(23) = 1; because the invalid instruction address test exceptional condition does not set the 'block I-Fetch' trigger, IC(23) = 1 if D(23) equalled 1 during the preceding branch end op.

The invalid instruction address test exceptional condition microprogram terminates with a normal end op. If no higher-priority exceptional condition or interruption is detected, the program interruption microprogram is entered next.

Q-Register Refill Exceptional Condition

- One extra I-Fetch cycle is performed to allow refilling of Q without conflicting with execution sequence of next instruction.

- Fetch sequencers are always activated.

The Q-register refill exceptional condition arises when Q-register (instruction buffer) refilling conflicts with the start of the next instruction. The exceptional condition delays processing of the next instruction by one (or two) cycles.

Following each storage request for instructions, the IC is incremented by 8 to obtain the address from which instructions will be fetched by the next request. This updating is accomplished by gating the contents of the IC to the parallel adder, adding a 1 to IC(20), and gating the incremented address back to the IC. The need to update the IC usually adds another cycle to the I-Fetch of RR, RX, RS, and SI instructions. There are two reasons for this extra cycle:

1. The SCI requires that each main storage address be retained for at least two cycles. Therefore, main storage requests during the first I-Fetch cycle would interfere with end-op requests.
2. Because the parallel adder is used to increment the IC, and the first cycle in the execution phase may also require the use of the parallel adder, the execution phase must be delayed until the IC is incremented.

The case when I-Fetch requires a second cycle (third if an indexed RX instruction) is treated as an exceptional condition in the CE. The 'EXCEP' micro-order in the first ROS word of the I-Fetch microprogram overrides the functional branch micro-order per the instruction op code [E(02–07)→ROA] and forces a new address into ROSAR. This forced address is determined by the format of the upcoming instruction and the status of the I-Fetch sequencers (Diagram 5-30, FEMDM). Operation of the I-Fetch sequencers is initiated at end op when the need to refill Q exists and the next instruction to be executed is not in the SS format. If the request was generated two cycles before end op, sequencer 2 is latched at the start of I-Fetch; otherwise, sequencer 1 is being set.

At the start of I-Fetch, The 'EXCEP' micro-order samples sequencer 1 to see if it is being set. If it is, a new address is always forced into ROSAR, causing one extra I-Fetch word to be added to the basic I-Fetch.

If sequencer 2 is found latched, the parallel adder is available for use on the next cycle because IC(20) has already been incremented. I-Fetch of RR and shift instructions does not require the fetching of an operand from main storage; also, storage operands for indexed RX instructions are not requested until the second basic I-Fetch cycle. For these reasons, sequencer 2 forces a new ROSAR address only if an RR, indexed RX, or shift instruction is not being fetched.

The forced ROSAR addresses as a result of the Q-register refill exceptional condition are shown in Table 3-1. The SS format is included for completeness. However, because the I-Fetch sequencers are not activated, a Q-register refill exceptional condition is never detected during the SS I-Fetch microprogram. Instead, the functions of the sequencers are initiated by micro-orders in that microprogram.

Two-Cycle RR I-Fetch

The actions of the first I-Fetch cycle are unchanged except for the overriding of the 'E(02–07)→ROA' micro-order by the 'EXCEP' micro-order (Diagram 5-6). During the second RR I-Fetch cycle, sequencer 2 is set and sequencer 1 is reset. This status increases the IC by 8 and returns the new instruction address to the IC at the start of the next cycle, thus completing the updating of IC(20). During the next cycle (first execution cycle), sequencers 3 and 1 are both set by the 'RASCR' micro-order in the forced word. This condition indicates that new instructions are to be gated to Q at the start of the second execution cycle. The major registers and timing applicable to this sequence are shown in Diagram 5-8, FEMDM.

Table 3-1. Q-Register Refill Exceptional Conditions

Instruction Being Fetched	Request Issued During Preceding:			Forced ROSAR Address (Hex)
	IC(21,22) at End Op	EEOP (2 Cycles Early)	NEOP, BEOP, or EEOP (1 Cycle Early)	
RR	00, 01, or 11	Never	Never	None
	10	Yes	Never	None
	10	No	Yes	030
RX, RS, or SI	00 or 11	Never	Never	None
Indexed RX, or shift RS	01 or 10	Yes	Never	None
Indexed RX	01 or 10	No	Yes	03A
Non-indexed RX	01 or 10	Yes	Never	022
Non-indexed RX	01 or 10	No	Yes	032
Shift RS	01 or 10	No	Yes	020
Non-shift RS, or SI	01 or 10	Yes	Never	024
Non-shift RS, or SI	01 or 10	No	Yes	034
SS	11	Never	Never	None
	00, 01, or 10	Never	Yes	None

†All shift instructions are of the RS format with an op code of 1000 1XXX.

Forced-Cycle RX I-Fetch

If the request to refill Q was not issued 2 cycles before end op, the actions of the second RX I-Fetch cycle include the same actions as the second RR I-Fetch cycle. Otherwise, IC(20) has already been incremented and sequencers 1 and 3 are automatically set during the second cycle. If the RX instruction is indexed, then a second forced word is now performed; this word contains the same micro-orders as the second word of the basic RX I-Fetch. Otherwise, the first forced word completes the I-Fetch routine after issuing any request inhibited during the first I-Fetch cycle. This action is performed by the 'MS-REQ*D-3' micro-order. D is also transferred to PAL if the request for new instructions was issued 2 cycles before end op. Because in this case sequencer 2 is reset during the first execution cycle, a 'SPEC' micro-order in the first execution word tests the storage address from PAL, not D. The new instructions are gated into Q [and the next op-code word to R from Q(0–15) if IC(21,22) = 10 at end op] at the start of the first execution cycle (second execution cycle if the request was not made two cycles early and the RX instruction is not indexed). The major registers and timing applicable to the non-indexed case are shown in Diagram 5-12, FEMDM.

Two-Cycle RS and SI I-Fetch

The major registers and timing applicable to the 2-cycle RS and SI I-Fetch are the same as that for the 2-cycle, non-indexed RX I-Fetch (Diagram 5-12). If the request to refill Q was not issued two cycles before end op, the actions of the second I-Fetch cycle include the same actions as the second RR I-Fetch cycle. Otherwise, IC(20) has already been incremented and sequencers 1 and 3 are automatically set during this forced cycle. This action results in the refill of Q at the start of the next cycle (first execution cycle).

Whichever cycle Q is refilled, Q(0–15) is also transferred to R if IC(21,22) was set to 10 during the preceding end op. This transferring of the op-code halfword is otherwise performed by an appropriate micro-order in the first I-Fetch word. If an MVI, STM, TS, I/O, or shift instruction is being fetched, no further I-Fetch actions are necessary. However, fetching of other RS or SI instructions causes the storage request omitted by the first I-Fetch cycle to be issued now. This request is performed by the 'MS-REQ*D-3' micro-order. Also, D is transferred to PAL if the Q-register refill request was issued two cycles before end op. Because in this case sequencer 2 is reset during the first execution cycle, any 'SPEC' micro-order tests the storage address from PAL, not D.

It was previously stated that the last I-Fetch word always includes the 'E(02–07)→ROA' micro-order. However, there is one exception to this statement: the forced I-Fetch cycle for shift instructions includes the 'E(04–07)→ROA' micro-order, which, in turn, forces the first execution cycle to branch to the second cycle per D rather than per PAL.

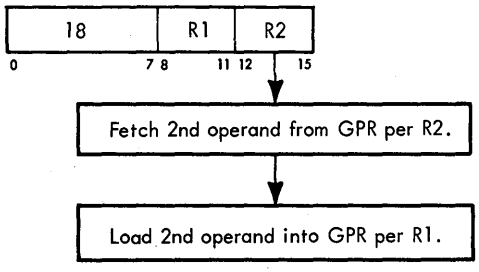
This section discusses the 35 instructions of the fixed-point instruction set. These instructions use the RR, RX, and RS formats. Positive fixed-point numbers are expressed in true binary form, whereas negative numbers are expressed in complement binary form (2's complement form). One operand is always in 1 of the 16 GPR's; the other operand may be in either a GPR or in main storage. For a discussion of number representation, data formats, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.

LOAD

The fixed-point load instructions provide a means of loading operands into the LS GPR's. The load operation may be register-to-register (RR format) or storage-to-register (RX and RS formats). In any case, the instruction loads the second operand into the first operand location, and the second operand location remains unchanged. In addition, certain load instructions can test the second operand before loading it and can load the second operand in complement, positive, or negative form.

Load, LR (18)

- Load 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:

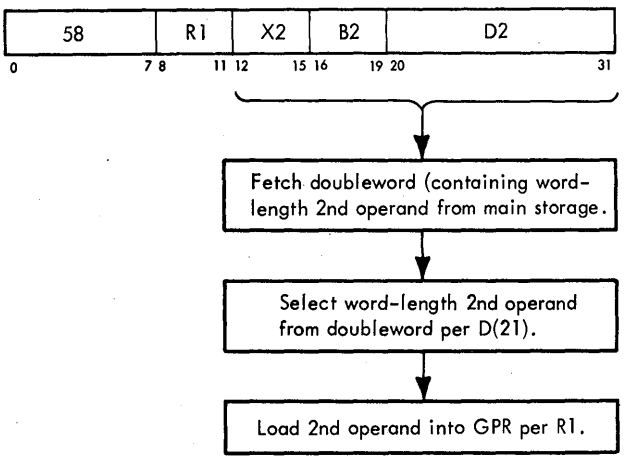


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.

The Load, LR, instruction loads the second operand from the GPR per R2 into the GPR per R1. At the start of execution, the word-length second operand is in S and T. Because both operands are in GPR's, no specification test is performed. The second operand is loaded into the GPR specified by R1, and an end-op cycle is taken.

Load, L (58)

- Load 2nd operand (in storage) into 1st operand location (in GPR per R1).
- RX format:

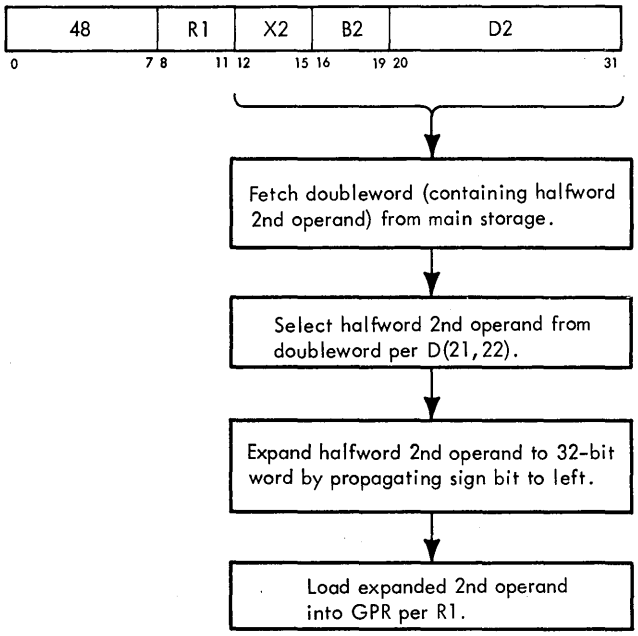


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword is to be stored: if a 1, right word; if a 0, left word.

The Load, L, instruction loads the second operand from main storage into the GPR per R1. Instruction execution starts with a specification test. If a program specification interruption occurs, an end op is forced and the instruction is suppressed. If no specification check exists, D(21) is tested to determine which word of the doubleword fetched from main storage will be gated from the SDBO to T. If D(21) = 1, the right word is gated; if D(21) = 0, the left word is gated. The contents of T are then loaded into the GPR specified by R1, and an end-op cycle is taken.

Load Halfword, LH (48)

- Load halfword 2nd operand (in storage) into 1st operand location (in GPR per R1).
- RX format:



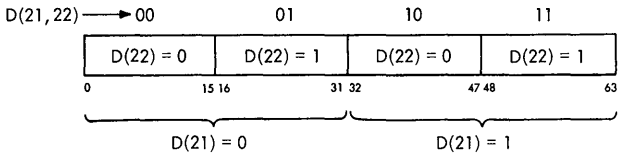
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- Halfword operand is expanded to word by propagating sign of halfword into 16 high-order bits of word-length register.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.

The Load Halfword, LH, instruction loads the halfword second operand (located in main storage) into the GPR specified by R1. The halfword obtained from main storage consists of 16 bits. Before loading the operand into LS, it is expanded to a 32-bit word by propagating the sign bit of the halfword through the 16 high-order bits of a word-length register.

Diagram 5-102, FEMDM, is a flowchart of the Load Halfword instruction. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and the first operand is in S and T. (The first operand plays no part in the instruction; it is subsequently destroyed when data is loaded into T.) During I-Fetch, a storage request was made to obtain the doubleword containing the halfword second operand.

The instruction first tests for a specification check condition. If a program specification interruption occurs, an end op is forced and the instruction is suppressed. If no specification check exists, the execution continues and may be divided into three general steps:

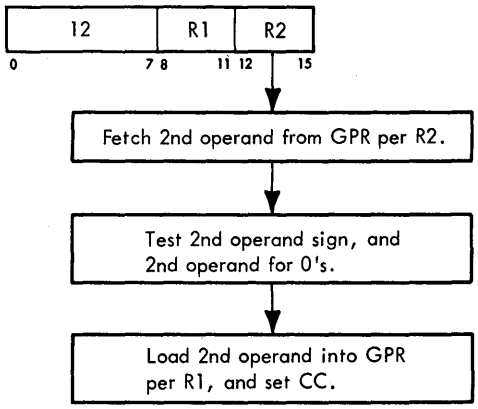
1. Set up A to propagate the sign:
   a. Load 1's into PAL(32–59), shift left four positions, and gate the result to A. ("A" now contains FFFF FF00.) The 1's are generated (ALD AP811) as a result of ROS word 0108 (CLD QB011) containing 100 in bit positions 82, 83, and 84 (CLD QZ011).
   b. Gate the contents of A to PAB, shift left four positions, and gate the result to A. ("A" now contains FFFF F000.)
   c. Gate the contents of A to PAB, shift left four positions, and gate the result to A. ("A" now contains FFFF 0000.)
2. Examine D(21) and D(22) to determine which halfword of the doubleword operand brought out from main storage is to be used as the halfword operand. The specified halfword may be in any one of four possible positions of the doubleword specified by D(21) and D(22):



   a. If D(21) = 0, gate SDBO(0–31) to T; if D(21) 1, gate SDBO(32–63) to T.
   b. If D(22) = 0, gate T(32–47) to PAA(48–63), and gate a 1 to PAA(47) if T(32) = 0. If D(22) = 1, gate T(48–63) to PAA(48–63), and gate a 1 to PAA(47) if T(48) = 0.
   c. Gate the contents of A to PAB(32–63).
3. Load the selected word:
   a. Gate PAL(32–63) to T.
   b. Gate the contents of T to the GPR specified by the R1 field in E(8–11).
   c. Take an end-op cycle.

Load and Test, LTR (12)

- Load 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1) and set CC according to result.
- RR format:



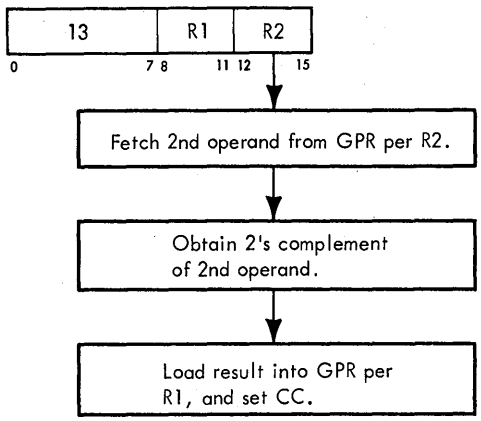
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- Test T(32) for plus or minus sign.
- STAT A and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Load and Test, LTR, instruction tests the second operand, from the GPR per R2, for all zeros and loads it into the GPR per R1. (If R1 and R2 specify the same GPR, the operation is equivalent to a test of the data without movement of the data.)

Diagram 5-103, FEMDM, is a flowchart of the instruction. The contents of T (second operand) are gated to PAA(32–63), and STAT A is set if PAL equals zero. The contents of T are then gated to the GPR specified by R1, and the CC is set as follows. If STAT A is set, the CC is set to 0. If STAT A is not set, the sign bit [T(32)] determines the CC: if the sign is minus [T(32) = 1], the CC is set to 1; if the sign is plus, the CC is set to 2. An end op completes instruction execution.

Load Complement, LCR (13)

- Load 2's complement of 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1) and set CC according to result.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT B if overflow occurred.
- Set STAT A if PAL is all 0's.
- Test T(32) for plus or minus sign.
- STAT's A and B, and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

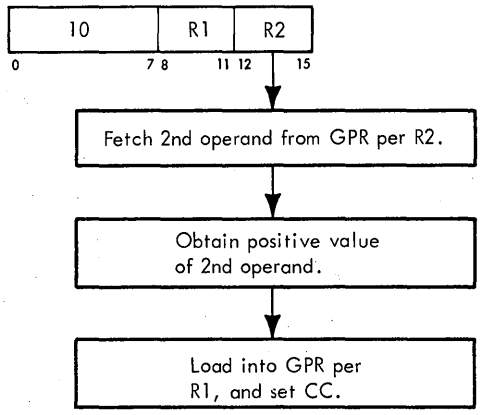
The Load Complement, LCR, instruction loads the 2's complement of the second operand from the GPR per R2 into the GPR per R1.

See Diagram 5-104, FEMDM, a flowchart of the instruction. The contents of T (second operand) are gated in 2's complement form to PAA(32–63). STAT A is set if PAL equals zero, and STAT B is set if a fixed-point overflow occurs. (Overflow occurs if the maximum negative number is 2's complemented.) The contents of PAL are transferred (via T) to the GPR specified by R1, and the CC is set as follows.

If STAT B is set, the CC is set to 3. If STAT A is set, the CC is set to 0. If neither STAT is set, the sign bit [T(32)] determines the CC: if the sign is minus [T(32) = 1], the CC is set to 1; if the sign is plus, the CC is set to 2. An end op completes instruction execution.

Load Positive, LPR (10)

- Load 2nd operand (unchanged if positive, 2's complemented if negative; in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- T(32) determines whether operand loaded is positive.
- If T(32) = 1, 2's complement operand.
- Set STAT B if overflow occurs.
- STAT's A and B and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Load Positive, LPR, instruction loads the absolute value of the contents of the GPR specified by R2 into the GPR specified by R1. The instruction also tests for an all-zero result and for an overflow condition. (Overflow occurs only when the maximum negative number is 2's complemented.) The results of the tests are indicated by the CC.

Diagram 5-105, FEMDM, is a flowchart of the Load Positive instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The first cycle of the instruction places the contents of T into PAL(32–63) and tests PAL for all 0's. If PAL equals zero, STAT A is set. The data in T is then loaded into the GPR per E(8–11) (R1).

Because the purpose of the instruction is to load only positive numbers, a test for negative numbers is made by examining T(32). If T(32) = 1, the data loaded in LS was a negative number. In this case, the contents of T must be converted to a positive number (true form) and reloaded into the GPR per E(8–11), thus destroying the negative number in that location.

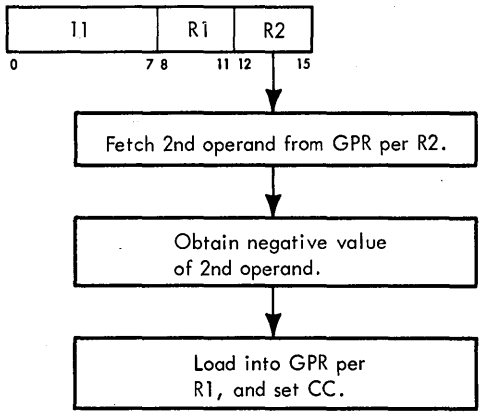
While in PAL, the 2's complement form of the data is tested to see whether overflow occurred when the number was converted. If overflow occurred, STAT B is set.

If T(32) = 0, the data loaded in LS was positive and need not be changed.

STAT's A and B, and T(32) determine the CC as follows. If STAT B is set, the CC is set to 3. If STAT A is set, the CC is set to 0. If neither STAT is set, the sign bit [T(32)] determines the CC; however, the sign can only be plus [T(32) = 0] and the CC is set to 2. An end-op cycle completes instruction execution.

Load Negative, LNR (11)

- Load 2nd operand (unchanged if negative, 2's complemented if positive; in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- T(32) determines whether operand loaded is negative.
- If T(32) = 0, 2's complement operand.
- STAT A and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.

The Load Negative, LNR, instruction loads the negative value of the contents of the GPR specified by R2 into the GPR specified by R1. The LNR instruction also tests the operand for all zeros and indicates the result in the CC.

See Diagram 5-106, FEMDM, a flowchart of the instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The first cycle of the instruction

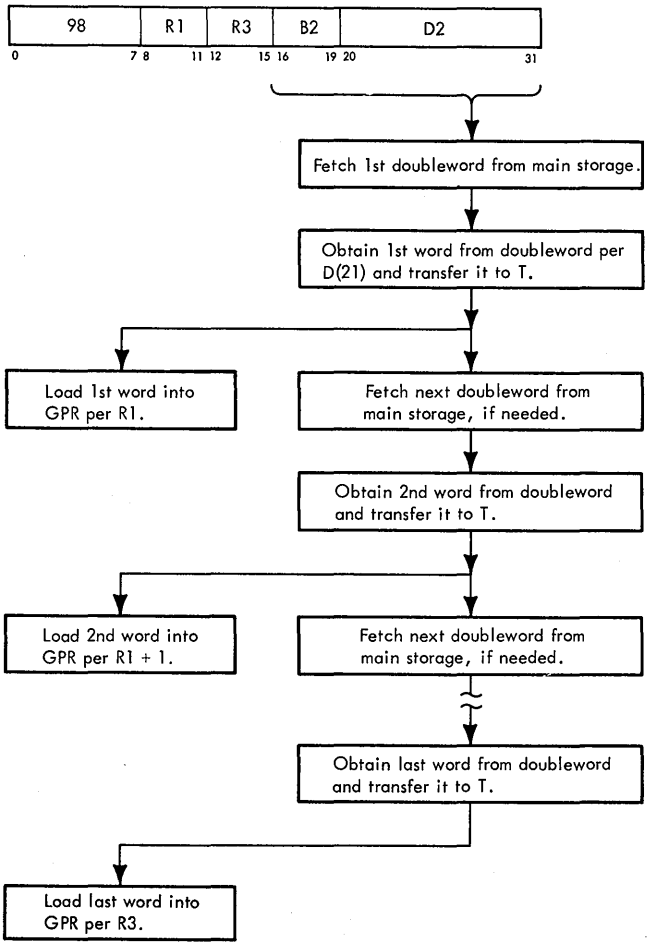
places the contents of T into PAA(32–63) and tests PAL for all 0's. If PAL contains all 0's, STAT A is set. The data in T, regardless of whether the result equals zero, is loaded into the GPR per E(8–11) (R1).

Because the purpose of the LNR instruction is to load only negative numbers, a test for positive numbers is made by examining T(32). If T(32) = 0, the data loaded into LS was a positive number. In this case, the contents of T must be converted to a negative number (2's complement form) and reloaded into the GPR per E(8–11), thus destroying the positive number in that location. If T(32) = 1, the data loaded into LS was a negative number and need not be changed.

STAT A and T(32) determine the CC as follows. If STAT A is set, the CC is set to 0. If STAT A is not set, the sign bit [T(32)] determines the CC; however, the sign can only be minus [T(32) = 1] and the CC is set to 1. An end-op cycle completes instruction execution.

Load Multiple, LM (98)

- Load 2nd operand (as many words as required; in storage) into GPR's, in ascending order, starting with 1st operand location (per R1) and ending with 3rd operand location (per R3).
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  Starting operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- R1 and R3 are compared to determine whether one or more words are to be loaded.
- If R1 = R3, only one word is to be loaded.
- If R3 is less than R1, addresses wraparound from GPR 15 to GPR 0.
- D(21) determines which word of doubleword is to be loaded into LS: if a 1, right word; if a 0, left word.
- 8 is added to D when more than one word is to be loaded.
- R1 is incremented by 1 each time a new word is to be loaded.

The Load Multiple, LM, instruction loads 32-bit words from main storage into LS. The GPR's are loaded in the ascending order of their addresses, starting with the GPR addressed by R1 and continuing up to and including the GPR addressed by R3. All combinations of GPR addresses specified by R1 and R3 are valid. When R3 is less than R1, the addresses wrap around from GPR 15 to GPR 0.

Diagram 5-107, FEMDM is a flowchart of the Load Multiple instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the starting operand address is in D. During I-Fetch, a storage request was made to fetch the operand addressed by D.

The instruction first tests for a specification check condition. If there is a specification check, a program specification interruption occurs and the operation is suppressed. Assuming there is no specification check, R1 [E(8–11)] and R3 [E(12–15)] are compared to determine whether one or more words are to be loaded. If R1 equals R3, only one word is to be loaded; if unequal, more than one word is to be loaded.

Assume one word is to be loaded (R1 = R3). D(21) is tested to determine which word of the doubleword from main storage is to be loaded. If D(21) is a 0, the left word is selected; if a 1, the right word is selected. The selected word is loaded into the GPR specified by R1, and an end-op cycle is taken, completing the instruction.

If more than one word is to be loaded (R1 and R3 are unequal), the main storage address in D is incremented by 8 to address the next doubleword in main storage, if it should be needed. D(21) is tested to determine which word of the doubleword from SDBO will be loaded first. R3 is decremented by 1, and R1 and R3 are again compared. As a result of these two tests, four possible conditions may exist, resulting in four different branches as follows:

1. If D(21) = 0 and R1 = R3, two words are to be loaded starting with the left word of the doubleword on the

SDBO. In this case, no further main storage request is necessary. SDBO(0–31) is gated to T, SDBO(0–63) is gated to AB, and the contents of T are loaded into the GPR specified by R1. E(8–11) (R1) is incremented by 1 to address the next sequential GPR, and the contents of B are transferred (via T) to the GPR specified by E(8–11) (R1 + 1).

2. If D(21) = 0 and R1 does not equal R3, more than two words are to be loaded starting with the left word. A main storage request per D is initiated to fetch the next doubleword from main storage. The microprogram enters the basic loop, which loads a doubleword, one word at a time, in consecutive GPR's and initiates another main storage request. An exit from the basic loop is made when one or two words remain to be loaded.
3. If D(21) = 1 and R1 = R3, two words are to be loaded starting with the right word. A main storage request per D is initiated because the second word to be loaded is contained in the next doubleword in main storage. SDBO(32–63) is transferred (via T) to the GPR specified by E(8–11) (R1). E(8–11) is incremented by 1, and, when the next doubleword is available, SDBO(0–31) is transferred (via T) to the GPR specified by E(8–11) (R1 + 1).
4. If D(21) = 1 and R1 does not equal R3, more than two words are to be loaded starting with the right word. A main storage request per D is initiated to fetch the next doubleword from main storage, and SDBO(32–63) is transferred (via T) to the GPR specified by E(8–11) (R1). E(8–11) is incremented by 1 to address the next sequential GPR, and D is incremented by 4 to address the next sequential doubleword in main storage. R1 and R3 are again compared; if equal, only two more words are to be loaded. When the requested doubleword is available, it is loaded, one word at a time, into the two sequential GPR's specified. If R1 does not equal R3, the microprogram enters the basic loop.

   Note that the last time D was incremented, it was incremented by 4 rather than by 8. At the start of this sequence, D(21) equalled 1, which is equivalent to a value of 4 in D. Adding 4 to D increases the value to 8, which will address the next sequential doubleword in main storage. D(21) has also been changed to a 0, which allows the microprogram to remain in the basic loop as long as is required. Note that a branch on D(21) is performed in the basic loop.

   The 4 that is added to D is developed in F. At the start of the instruction, F was set to -64 (1100 0000). F(0) is then set to 0, establishing a value of 0100 0000 in F. F(0–3) and F(4–7) are transposed, giving a value of 0000 0100 (4) in F. When F(4–7) is added to D, D is incremented by 4.

   When the last word has been loaded into the GPR specified by R3, an end-op cycle is taken, completing the instruction.

ADD-TYPE INSTRUCTIONS

- Fixed-point add-type instructions use RR and RX formats.
- 2nd operand is algebraically added to 1st operand.
- For subtract and compare instructions, 2nd operand is in 2's complement form.
- Except for compare instructions, result is stored into 1st operand location.
- CC is determined by op code and hardware conditions.

The fixed-point add-type instructions use the RR format with word-length operands, the RX format with word-length operands, and the RX format with a halfword second operand.

At the start of execution of RR format fixed-point instructions, the first operand is in B (also in A and D) and the second operand is in T (also in S).

At the start of execution of RX format fixed-point instructions, the first operand is in S and T, and a main storage request for the second operand has been issued per D. Because the second operand is fetched from main storage, a specification test is performed (Diagram 5-108, Sheet 1, FEMDM), and a program specification interruption is taken if the second operand address does not specify integral boundaries. If a program specification interruption is taken, the instruction is suppressed. If no specification check occurs, the first operand is transferred from T to B, and the specified word of the doubleword requested from main storage is selected per D(21) and is gated to T. The first operand is now in B and the second operand is in T, which is the same condition which would exist after an RR I-Fetch.

If the RX format instruction specifies a halfword second operand, two additional functions must be performed. The desired halfword [selected per D(22)] of the word in T [selected per D(21)] must be loaded into the low-order halfword of T, and the sign bit must be propagated left to fill the high-order halfword of T.

The fixed-point add-type instructions may be divided into three functional groups: add, subtract, and compare. All add-type instructions set a CC, and all except compare instructions store the result into the first operand location.

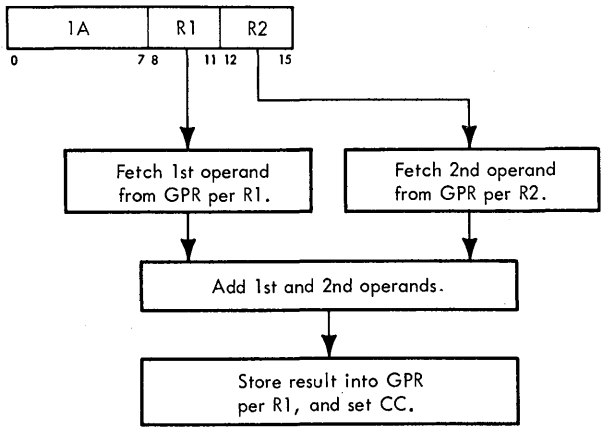
The CE performs fixed-point add-type instructions as follows (Diagram 5-108, Sheet 2):

1. The second operand is algebraically added to the first operand and the result is stored (except for compare instructions) into the first operand location. For subtract and compare instructions, the second operand (sign bit and integer) is 2's complemented, which, in effect, inverts the sign.
2. Because of the sign notation used, and because positive numbers exist in true binary form and negative numbers exist in 2's complement form, the operand signs are treated as high-order extensions of the integers.

3. Except for Add Logical and Subtract Logical instructions, the sign bit of the result [T(32)] is used as one factor in determining the CC; carry conditions from the high-order digit and from the sign bit are tested for a fixed-point overflow condition (recorded in STAT B).
4. For all fixed-point add-type instructions, a zero result is indicated by setting STAT A.
5. For Add Logical and Subtract Logical instructions, the sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition to determine the CC. The result of Add Logical or Subtract Logical instructions is the same as for the corresponding add or subtract instruction, except that the result is not tested for a fixed-point overflow condition and the significance of the CC is different. (See Table in Sheet 2 of Diagram 5-108.)

Add, AR (1A)

- Algebraically add 2nd operand (in GPR, per R2) to 1st operand (in GPR, per R1) and place result into 1st operand location.
- RR format:

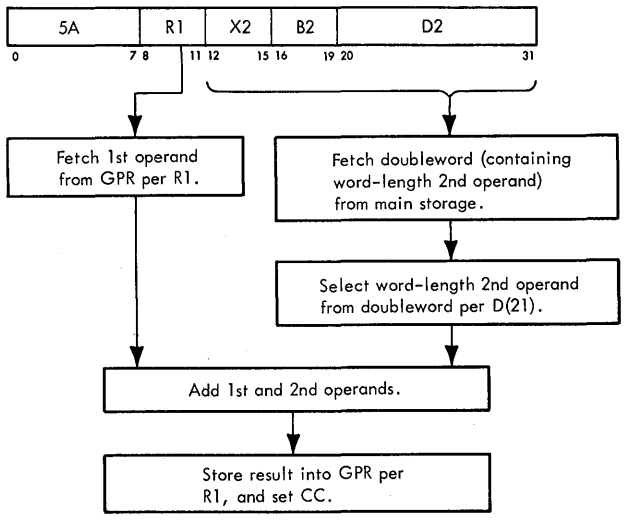


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset] : CC = 1.
  Result is greater than zero [T(32) = 0, and STAT's A and B are reset] : CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add, AR, instruction algebraically adds the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add, A (5A)

- Algebraically add 2nd operand (in storage) to 1st operand (in GPR, per R1) and place result into 1st operand location.
- RX format:



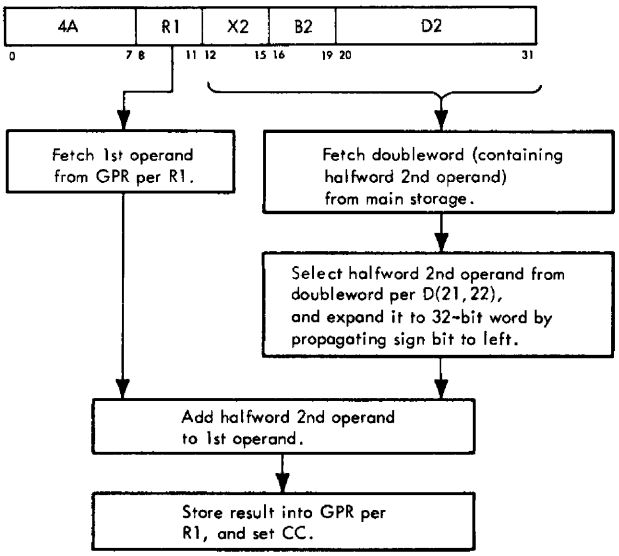
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset] : CC = 1.
  Result is greater than zero [T(32) = 0, and STAT's A and B are reset] : CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add, A, instruction algebraically adds the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the word-length second operand: if D(21) = 1, the right word; if D(21) = 0, the left word. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Halfword, AH (4A)

- Algebraically add halfword 2nd operand (in storage) to 1st operand (in GPR per R1) and place result into 1st operand location.

- RX format:



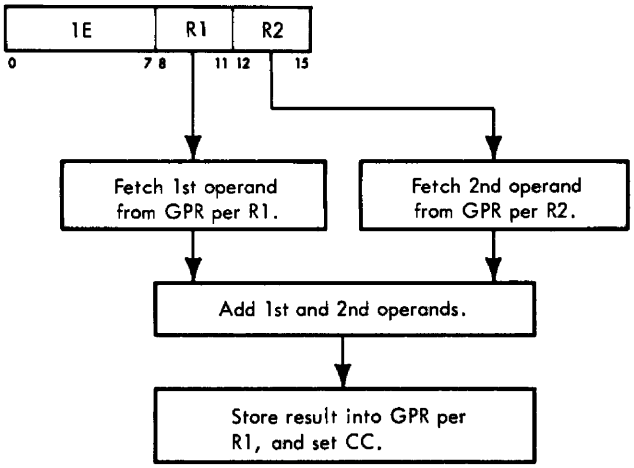
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset] : CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset] : CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add Halfword, AH, instruction algebraically adds the halfword second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the halfword second operand, and D(22) determines which halfword of that word contains the second operand, as follows: D(21) = 0, left word; D(21) = 1, right word; D(22) = 0, left halfword; D(22) = 1, right halfword. When the halfword second operand is selected, it is expanded to a word by propagating the sign bit through the 16 high-order bit positions of T.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Logical, ALR (1E)

- Algebraically add 2nd operand (in GPR per R2) to 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:



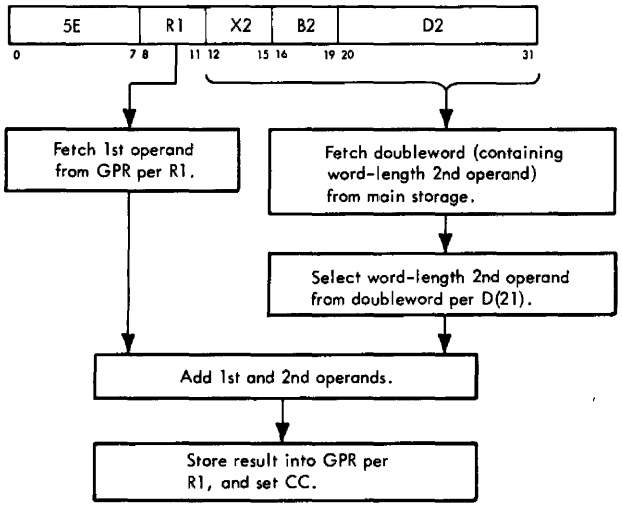
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero and no carry from PAL(32) [STAT A is set and A(31) = 0] : CC = 0.
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

The Add Logical, ALR, instruction algebraically adds the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the sum is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The sum is the same as for the AR instruction; the only difference in execution is that the sum is not tested for a fixed-point overflow condition, and that the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Logical, AL (5E)

- Algebraically add 2nd operand (in storage) to 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero and no carry from PAL(32) [STAT A is set and A(31) = 0] : CC = 0.
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

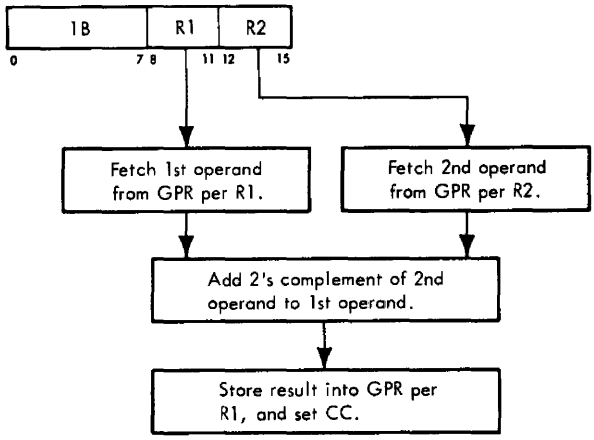
The Add Logical, AL, instruction algebraically adds the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the word-length second operand: if D(21) = 1, the right word; if D(21) = 0, the left word.

The sign bit of the sum is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The sum is the same as for the A instruction; the only difference in execution is that the sum is not tested for a fixed-point overflow condition, and that the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract, SR (1B)

- Algebraically subtract 2nd operand (in GPR, per R2) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:



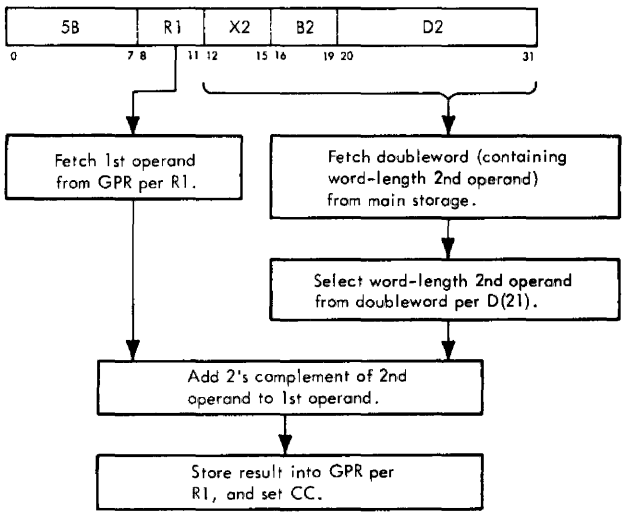
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset] : CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset] : CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract, SR, instruction adds the 2's complement of the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the SR and AR instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract, S (5B)

- Algebraically subtract 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



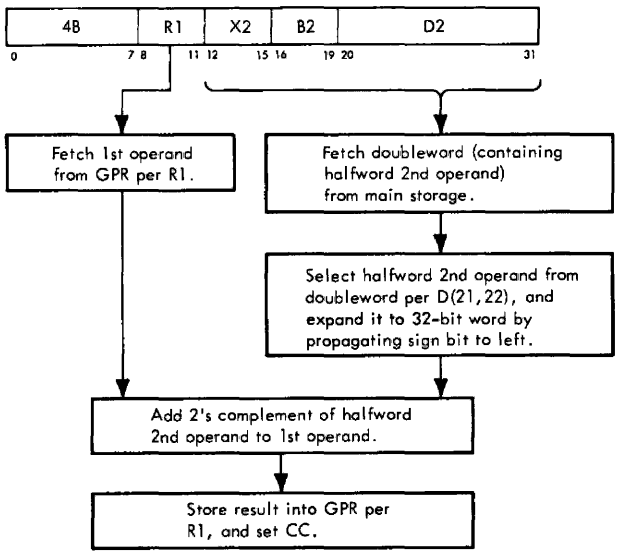
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract, S, instruction adds the 2's complement of the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the S and A instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Halfword, SH (4B)

- Algebraically subtract halfword 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

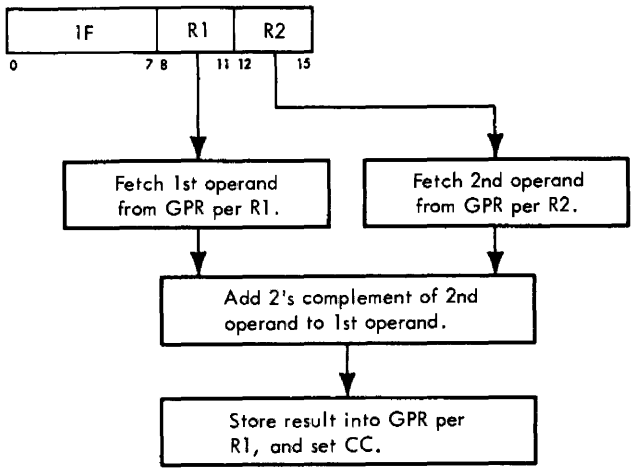
The Subtract Halfword, SH, instruction adds the 2's complement of the halfword second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the SH and AH instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Logical, SLR (1F)

- Algebraically subtract 2nd operand (in GPR per R2) from 1st operand (in GPR per R1) and place result into 1st operand location.

- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.

- CC setting:
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

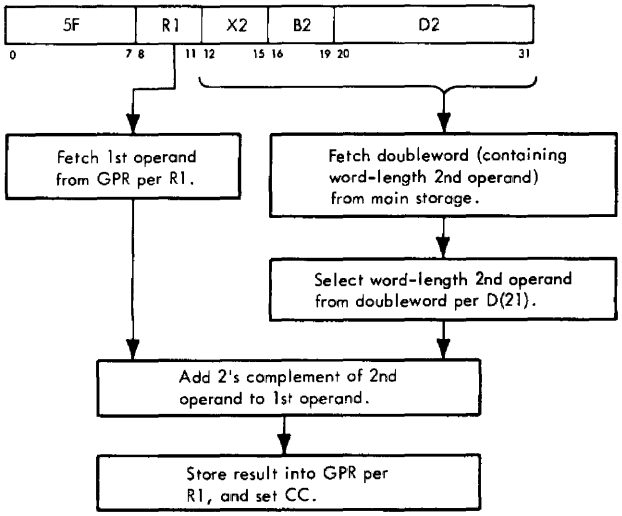
The Subtract Logical, SLR, instruction adds the 2's complement of the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The result is the same as for the ALR instruction; the difference in execution is that the second operand is 2's complemented, the result is not tested for a fixed-point overflow condition, and the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Logical, SL (5F)

- Algebraically subtract 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.

- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.

- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.

- CC setting:
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

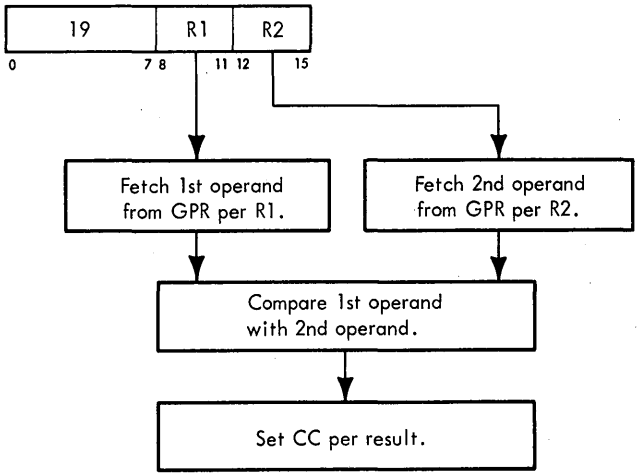
The Subtract Logical, SL, instruction algebraically adds the 2's complement of the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The result is the same as for the AL instruction; the difference in execution is that the second operand is 2's complemented, the result is not tested for a fixed-point overflow condition, and the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare, CR (19)

- Algebraically compare 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and set CC according to result.

- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.

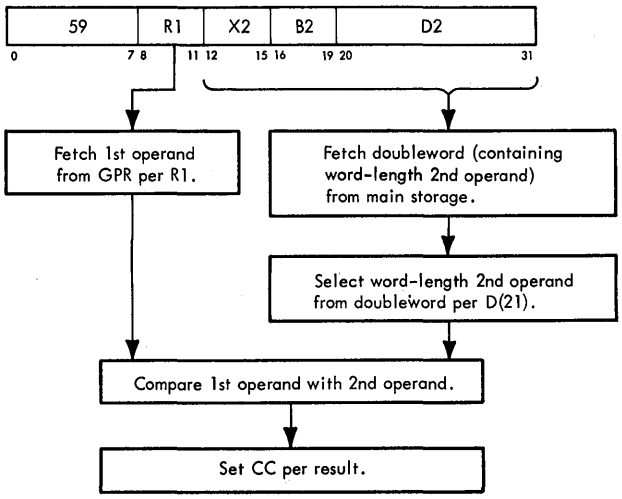
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1] : CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0] : CC = 2.

The Compare, CR, instruction algebraically compares the first operand (from the GPR per R1) with the second operand (from the GPR per R2) and sets the CC according to the result. The compare operation is accomplished by adding the 2's complement of the second operand to the first operand and setting the CC according to the result. The result is not stored. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare, C (59)

- Algebraically compare 1st operand (in GPR per R1) with 2nd operand (in storage) and set CC according to result.

- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.

- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.

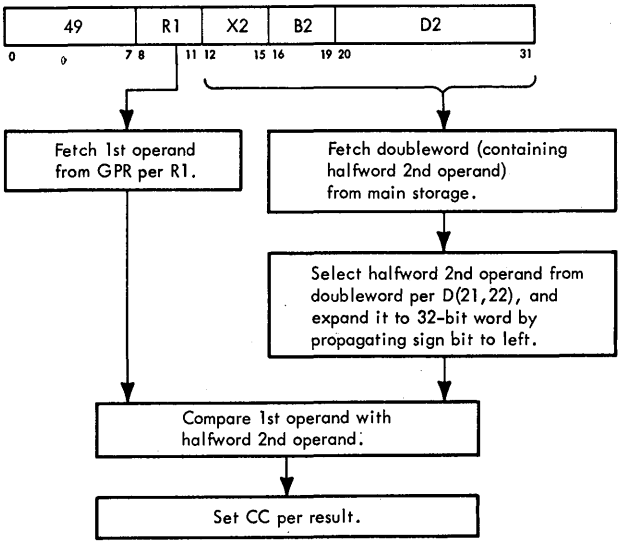
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1] : CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0] : CC = 2.

The Compare, C, instruction algebraically compares the first operand (from the GPR per R1) with the second operand (from storage) and sets the CC according to the result.

Because the word-length second operand is in main storage, D(21) determines which word of the doubleword fetched from main storage contains the second operand: if a 1, right word; if a 0, left word. The compare operation is accomplished by adding the 2's complement of the second operand to the first operand and setting the CC according to the result. The result is not stored. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare Halfword, CH (49)

- Algebraically compare 1st operand (in GPR per R1) with halfword 2nd operand (in storage) and set CC according to result.
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1] : CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0] : CC = 2.

The Compare Halfword, CH, instruction algebraically compares the first operand (from the GPR per R1) with the halfword second operand (from storage) and sets the CC according to the result.

Because the halfword second operand is in main storage, D(21) determines which word of the doubleword fetched from main storage contains the halfword second operand: if a 1, right word; if a 0, left word. D(22) determines which half of that word contains the second operand: if a 1, right half; if a 0, left half. The halfword second operand is expanded to word-length by propagating the sign bit through the high-order 16-bit positions of T. The compare operation is accomplished by adding the 2's complement of the halfword second operand to the first operand and setting the CC according to the result. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

MULTIPLY

There are three fixed-point multiply instructions:

1. Multiply, MR, RR format – which uses a 32-bit multiplier and multiplicand, and produces a 64-bit product.
2. Multiply, M, RX format – which uses a 32-bit multiplier and multiplicand, and produces a 64-bit product.
3. Multiply Halfword, MH, RX format – which uses a 16-bit multiplier and a 16-bit multiplicand, and produces a 32-bit product.

Note: In the Multiply, M, and Multiply Halfword, MH, instructions, the second operand is the multiplicand and the first operand is the multiplier.

Each of the three fixed-point multiply instructions has a unique initialization routine. The initialization routines (Diagram 5-109, Sheet 1, FEMDM) perform a specification test, set E(12–15) to 15, set the STC to 3, and establish the operands in S and T as follows:

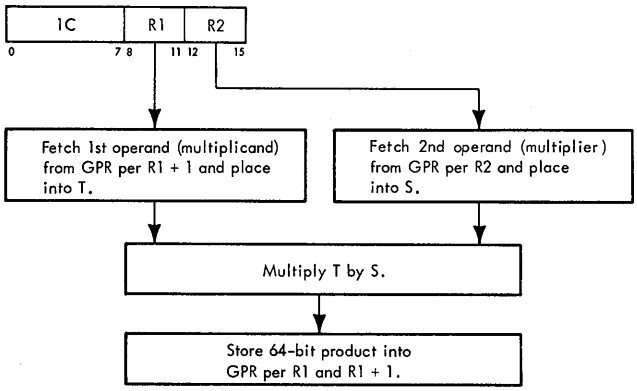
1. Multiply, MR, RR format – Transfers the multiplier (second operand from GPR per R2) to S and the multiplicand (first operand from GPR per R1 + 1) to T.
2. Multiply, M, RX format – Transfers the multiplicand (first operand from GPR per R1 + 1) to S and the multiplier (second operand from main storage) to T.
3. Multiply Halfword, MH, RX format – Transfers the multiplicand (first operand from GPR per R1) to S and the multiplier (halfword second operand from main storage, expanded to word length by propagating the sign bit through the 16 high-order bit positions) to T.

A common multiply microprogram is then entered. Multiples of T are selected per bit-pairs from S and are added to a partial product in B to form a new partial product. Low-order partial product bit-pairs are accumulated in F and SAL. When SAL has accumulated a partial product byte, it is stored into S, replacing the byte of S already used. Sixteen multiply cycles are taken until a

word (four bytes) of product is accumulated in S. PAL now contains the high-order word of the product and S contains the low-order word of the product. The product is stored in an even/odd pair of GPR's specified by R1 and R1 + 1, and an end-op cycle is taken to terminate the operation. (For the Multiply Halfword instruction, only the low-order 32 bits of the product are stored into the GPR per R1.)

Multiply, MR (1C)

- Multiply 1st operand (in GPR per R1 + 1) by 2nd operand (in GPR per R2) and place 64-bit product into 1st operand location (in GPR per R1 and R1 + 1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  Contents of even-address GPR specified by R1 are in A, B, and D (not used).
  Multiplicand (1st operand) is in odd-address GPR specified by R1 + 1.
  2nd operand (multiplier) is in S and T.
- Multiple selection bits (M1,M2) are selected from multiplier (in S) per E(12–15).
- Multiples of multiplicand (in T) are selected by M1,M2 bits and 'TX' trigger.
- Multiples of multiplicand are added to partial product in B.
- Partial product bits from B(66,67) are accumulated in SAL and F per E(14,15).
- SAL contains byte of partial product when filled.
- SAL is transferred to correct byte in S per STC.
- When last (4th) byte is transferred to S, multiplier in S is replaced by low-order half of product; high-order bits are in PAL.

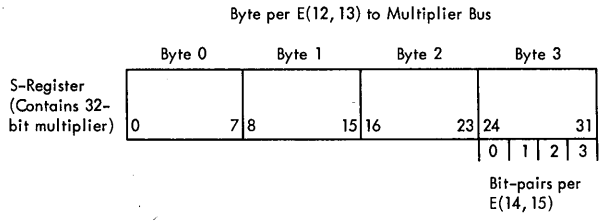
The Multiply, MR, instruction multiplies the contents of T (multiplicand) by the contents of S (multiplier). Because both the multiplier and the multiplicand are 32-bit signed integers, the product is a 64-bit signed integer and must be stored into an even/odd pair of GPR's.

A flowchart of the operation is shown in Diagram 5-109, FEMDM. To correctly specify the even/odd GPR pair, the R1 field of the instruction must refer to an even-address GPR or a program specification interruption occurs. After R1 is tested to see whether it is even, 15 and 3 are placed into E(12–15) and the STC, respectively. The value in E(12–15) selects the correct multiple selection bits (M1,M2) from the multiplier in S, and the value in the STC correctly positions the partial product byte in S. Each value is sequentially reduced during the operation. The value in T is now destroyed, and the multiplicand is transferred from the GPR per R1 + 1 to T. At this point, S contains the multiplier and T contains the multiplicand.

Execution of the MR instruction occurs in three iterative steps.

1. Selection of multiplicand multiples.
2. Addition of multiplicand multiples to partial product to form a new partial product.
3. Extraction of partial product bits to form a product.

Multiple selection bits (M1,M2) are selected from S per E(12–15), which is initially set to 15 (decimal) and is decremented by 1 each time multiple selection bits are selected. E(12,13) determines which byte of S is gated to the multiplier (MPR) bus, and E(14,15) determines which bit-pair of the selected byte is used to set M1,M2:



Bits M1,M2, which have the same bit configuration as the bit-pair selected from S by E(12–15), are used with the 'TX' trigger to select a multiple of the multiplicand which will be added to a partial product to develop a new partial product. The multiple selected for all combinations of M1,M2 bits and the state of the 'TX' trigger is listed in Table 3-2.

Four bit configurations of M1,M2 are possible, representing decimal values of 0, 1, 2, and 3. Five multiplicand multiples can be selected: 0 x T, 1 x T, -1 x T, 2 x T and -2 x T. Multiplicand multiples are developed and applied as follows:

1. 0 x T: Zero's are added to the partial product.

2. 1 x T: The multiplicand (in T) is added to the partial product.
3. -1 x T: The multiplicand is added in 2's complement form to the partial product.
4. 2 x T: The multiplicand is gated to PAA, shifted left one bit position (in effect, doubles its value), and added to the partial product.
5. -2 x T: The multiplicand, shifted left one bit position, is added in 2's complement form to the partial product.

No provision has been made to develop a multiple of the multiplicand of 3 x T. Therefore, when M1,M2 has a decimal value of 3, a multiple of -1 x T is selected and the 'TX' trigger is set and remains set into the next multiply cycle. Note in Table 3-2 that when the 'TX' trigger is set during the selection of a multiple, it has the effect of increasing the value of the multiple by 1 for the corresponding value of M1,M2. Because the partial product is shifted right two positions before each multiple is added, the value of the multiple is increased by a factor of 4. The effect of the 'TX' trigger's being set is to increase the value of the multiple (defined by M1,M2) by 4. Thus the multiplicand is, in effect, multiplied by -1 and +4 (plus the multiple which would have been selected if the 'TX' trigger were not set).

When the partial product is shifted right two positions (right 4 and left 2) after each addition of the selected multiple of the multiplicand, the low-order bit-pair of the partial product is gated to SAL per E(14,15) and is added to the contents of F. When a byte of the partial product is accumulated in F, the byte is transferred to S per the STC (via SAL), replacing the byte of multiplier which has been used.

When the low-order partial product bit-pair is gated to SAL, E(12–15) has been decremented twice and the low-order bit pair is stored into F(6,7) per the value of 01 in E(14,15) (C of Sheet 3, Diagram 5-109, FEMDM).

As previously mentioned, the partial product bits are transferred via SAL to F. This register accumulates a partial product byte (eight bits). When the last two partial product bits, required to complete a byte, are selected [E(14,15) = 10], the contents of F are transferred to SAL, where the two partial product bits are positioned correctly. This byte is then transferred to S and positioned according to the value of the STC. At this point, the STC is equal to 011, thus placing the partial product byte into S(24–31). The STC and E(12–15) are then reduced by 1, and selection of partial product bits is continued. The microprogram remains in the multiple selection loop until E(12,13) = 00 when tested. At this time, E(12–15) contains 0001 and is decremented to 0000; and the microprogram enters the multiply termination routine.

During the multiply termination routine, five events take place:

1. A multiple is selected for the high-order bit-pair in S.
2. Because there is a 2-cycle lag between the selection of a multiple and the storage of the corresponding partial product bit-pair into F, E(12–15) wraps around to 1110 to control the storage of the last two bit-pairs of the partial product in F (via SAL).
3. The high-order partial product byte is transferred from F to S per the STC. (The last bit-pair, however, does not go to F, because the last byte is gated directly from SAL to S.)
4. The high-order word of the partial product is transferred from PAL to T, and from T to the GPR specified by R1.
5. The low-order word of the partial product in S is transferred to T, and from T to the GPR specified by R1 + 1.

This sequence places the complete product into the even/odd GPR pair in LS. An end-op cycle is then taken, and the operation is finished.

The product sign follows the rules of algebra (except that the sign of a zero product is plus); however, the sign bits are manipulated as though they were high-order extensions of the integer throughout the multiply operation. Two multiply examples follow; the first, in Figure 3-6, uses two positive operands, and the second, in Figure 3-7, uses the same operands with minus signs. Note that the product is the same in both cases with no special handling of the sign bits required.

Table 3-2. Value of Multiple Determined by Multiple Selection Bits (Fixed-Point)

Multiple Selection Bits		'TX' Trigger	T-Register Times Value Indicated	Set 'TX' Trigger
M1	M2			
0	0	0	0 x T	No
0	1	0	1 x T	No
1	0	0	2 x T	No
1	0	0	-2 x T	Not
1	1	0	-1 x T (2's Complement)	Yes
0	0	1	1 x T	No
0	1	1	2 x T	No
1	0	1	-1 x T (2's Complement)	Yes
1	1	1	0 x T	Yes

† Used on last multiple selection if multiplicand is negative.

	Hex	Binary	Decimal
Multiplicand (in T)	001A	0000 0000 0001 1010	+26
Multiplier (in S)	00A3	0000 0000 1010 0011	x +163
Product (to S via F)	108E	0001 0000 1000 1110	+4238

1. M1, M2 bits are derived from S per E(12-15).
2. See Table 3-2 for multiple selection.

M1, M2	'TX' Tgr	Multiple	E(12-15)	Remarks	Contents in PAL	Accumulate Bit-Pairs in F; Transfer Bytes to S
	0		1111	Initial partial product	0000 0000 0000 0000	
1 1	0	-1 x T	1111	+ T (2's complement)	1111 1111 1110 0110	
	1		1110	New partial product	1111 1111 1110 0110	
	1		1110	Right 2 positions	1111 1111 1111 1001	10
0 0	1	1 x T	1110	+ T (true)	0000 0000 0001 1010	10
	0		1101	New partial product	0000 0000 0001 0011	
	0		1101	Right 2 positions	0000 0000 0000 0100	11
1 0	0	2 x T	1101	+ 2T (true, left 1)	0000 0000 0011 0100	1110
	0		1100	New partial product	0000 0000 0011 1000	
	0		1100	Right 2 positions	0000 0000 0000 1110	00
1 0	0	2 x T	1100	+ 2T (true, left 1)	0000 0000 0011 0100	00 1110
	0		1011	New partial product	0000 0000 0100 0010	
	0		1011	Right 2 positions	0000 0000 0001 0000	10
0 0	0	0 x T	1011	Add 0's	0000 0000 0000 0000	1000 1110
	0		1010	New partial product	0000 0000 0001 0000	
	0		1010	Right 2 positions	0000 0000 0000 0100	00
0 0	0	0 x T	1010	Add 0's	0000 0000 0000 0000	00
	0		1001	New partial product	0000 0000 0000 0100	
	0		1001	Right 2 positions	0000 0000 0000 0001	00
0 0	0	0 x T	1001	Add 0's	0000 0000 0000 0000	0000
	0		1000	New partial product	0000 0000 0000 0001	
	0		1000	Right 2 positions	0000 0000 0000 0000	01
0 0	0	0 x T	1000	Add 0's	0000 0000 0000 0000	01 0000
	0		0111	New partial product	0000 0000 0000 0000	
	0		0111	Right 2 positions	0000 0000 0000 0000	00
0 0	0	0 x T	0111	Add 0's	0000 0000 0000 0000	0001 0000
						S ← 0001 0000 1000 1110
						1 0 8 E

and so on

Notes:
1. T = T-register.
2. For RX format instruction, reverse multiplier and multiplicand.

Figure 3-6. Fixed-Point Multiply, Example No. 1 (RR Format)

	Hex	Binary	Decimal
Multiplicand (in T)	FFE6	1111 1111 1110 0110	-26
Multiplier (in S)	FF5D	1111 1111 0101 1101	x -163
Product (to S via F)	108E	0001 0000 1000 1110	+4238

1. M1, M2 bits are derived from S per E(12-15).
2. See Table 3-2 for multiple selection.

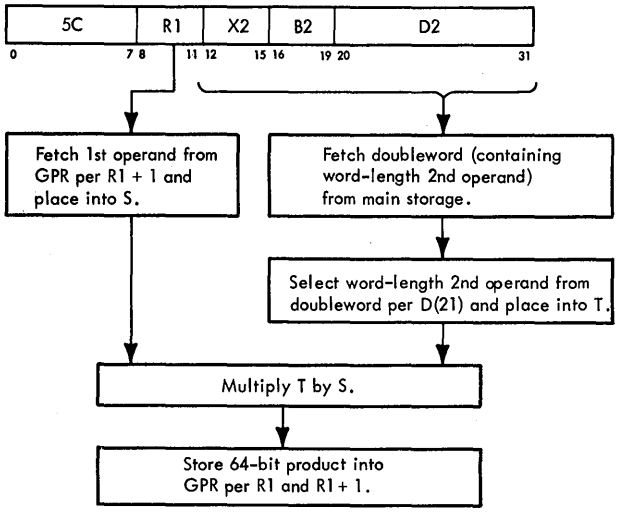
M1, M2	'TX' Tgr	Multiple	E(12-15)	Remarks	Contents in PAL	Accumulate Bit-Pairs in F; Transfer Bytes to S
	0		1111	Initial partial product	0000 0000 0000 0000	
0 1	0	1 x T	1111	+ T (true)	1111 1111 1110 0110	
	0		1110	New partial product	1111 1111 1110 0110	
	0		1110	Right 2 positions	1111 1111 1111 1001	10 → 10
1 1	0	-1 x T	1110	+ T (2's complement)	0000 0000 0001 1010	
	1		1101	New partial product	0000 0000 0001 0011	
	1		1101	Right 2 positions	0000 0000 0000 0100	11 → 1110
0 1	1	2 x T	1101	+ 2T (true, left 1)	1111 1111 1100 1100	
	0		1100	New partial product	1111 1111 1101 0000	
	0		1100	Right 2 positions	1111 1111 1111 0100	00 → 00 1110
0 1	0	1 x T	1100	+ T (true)	1111 1111 1110 0110	
	0		1011	New partial product	1111 1111 1101 1010	
	0		1011	Right 2 positions	1111 1111 1111 0110	10 → 1000 1110
1 1	0	-1 x T	1011	+ T (2's complement)	0000 0000 0001 1010	
	1		1010	New partial product	0000 0000 0001 0000	
	1		1010	Right 2 positions	0000 0000 0000 0100	00 → 00
1 1	1	0 x T	1010	Add 0's	0000 0000 0000 0000	
	1		1001	New partial product	0000 0000 0000 0100	
	1		1001	Right 2 positions	0000 0000 0000 0001	00 → 0000
1 1	1	0 x T	1001	Add 0's	0000 0000 0000 0000	
	1		1000	New partial product	0000 0000 0000 0001	
	1		1000	Right 2 positions	0000 0000 0000 0000	01 → 01 0000
1 1	1	0 x T	1000	Add 0's	0000 0000 0000 0000	
	1		0111	New partial product	0000 0000 0000 0000	
	1		0111	Right 2 positions	0000 0000 0000 0000	00 → 0001 0000
1 1	1	0 x T	0111	Add 0's	0000 0000 0000 0000	
			≈			S ← 0001 0000 1000 1110
			and so on			1 0 8 E

Notes:
1. T = T-register.
2. For RX format instructions, reverse multiplier and multiplicand.

Figure 3-7. Fixed-Point Multiply, Example No. 2 (RR Format)

Multiply, M (5C)

- Multiply 1st operand (in GPR per R1 + 1) and 2nd operand (in storage) and place 64-bit result into 1st operand location (in GPR per R1 and R1 + 1).
- See Note under "Multiply."
- RX format:

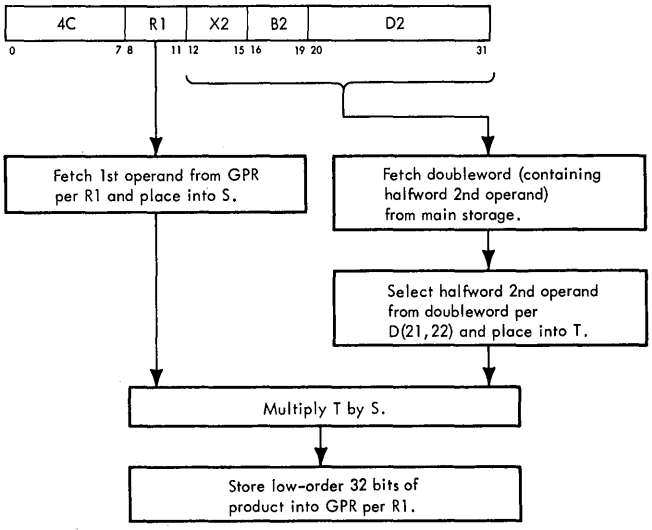


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Contents of even-address GPR per R1 are in S and T (not used).
  1st operand is in odd-address GPR per R1 + 1.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.

The Multiply, M, instruction multiplies the contents of T (second operand from storage) and the contents of S (first operand, from GPR per R1 + 1), and stores the 64-bit product into the GPR's per R1 and R1 + 1. Once the operands have been obtained, the operation is identical to that of the Multiply, MR, instruction, except that the roles of the multiplier and multiplicand are reversed. (See Note under "Multiply" and Diagram 5-109.)

Multiply Halfword, MH (4C)

- Multiply 1st operand (in GPR per R1) and halfword 2nd operand (in storage) and place low-order 32 bits of result into 1st operand location.
- See Note under "Multiply".
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.

The Multiply Halfword, MH, instruction multiplies the contents of T (expanded halfword second operand from main storage) and the contents of S (first operand from GPR per R1) and stores the 32 low-order bits of the product into the first operand location. D(21,22) determines the location of the second operand within the doubleword obtained from main storage. The second operand is then expanded to a word-length operand by propagating the sign bit through the high-order 16 bit positions of T.

From this point, the operation is identical to that of the Multiply, MR, instruction, except that the roles of the multiplier and multiplicand are reversed. (See Note under "Multiply.") For a flowchart of the operation, see Diagram 5-109.

DIVIDE

Fixed point division is performed by repetitive reduction of the dividend by multiples of the divisor to obtain a

remainder whose value is less than that of the divisor, and to accumulate partial quotient (PQ) bits derived from the partial remainders to form a quotient. The basic nonrestoring method is used, with the dividend in true form. Nonrestoring division means that if a negative remainder is obtained during the reduction cycles (an overdraw has been made), the remainder is not corrected, but instead the next divisor multiples are made positive until the remainder becomes positive again. (In most valid divide operations, the first reduction cycle is an intentional overdraw.)

There are two fixed-point divide instructions: Divide, DR, RR format, and Divide, D, RX format. Each has a unique initialization routine. The initialization routines: provide a specification test; place the low-order half of the dividend into S and the high-order half of the dividend into B, in true form regardless of the sign; place the divisor into T; store the signs for later use; and set E(12–15) and the STC to 0.

A common divide microprogram is then entered. Two bits of the low-order dividend are appended to the high-order dividend. A multiple of the divisor is selected to reduce the dividend to a partial remainder. The inverted sign of the partial remainder is stored into F as a partial quotient (PQ) bit. Thirty-two such reduction cycles are taken, accumulating PQ bits in F until a byte of quotient is obtained. The quotient byte is stored into S, replacing the byte of the low-order dividend which has been used. When four quotient bytes have been stored into S, S contains the quotient and B contains the remainder.

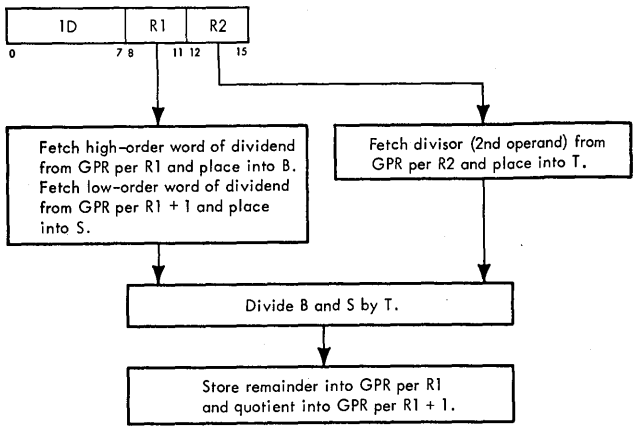
The microprogram now enters one of four termination routines, determined by the sign of the divisor and the form (true or 2's complement) of the quotient. The termination routines establish the proper sign and form of the remainder and quotient, according to the convention of fixed-point arithmetic and the rules of algebra. The remainder and quotient are then stored in an even/odd pair of GPR's specified by R1 and R1 + 1, and an end-op cycle is taken to terminate the operation.

Divide, DR (1D)

- Divide 1st operand (in GPR per R1 and R1 + 1) by 2nd operand (in GPR per R2) and place result into 1st operand location (remainder in GPR per R1; quotient in GPR per R1 + 1).

- RR format (See adjoining column.)

- Conditions at start of execution:
  Instruction is in E.
  High-order half of dividend (1st operand) is in A, B, and D.
  Low-order half of dividend is in GPR per R1 + 1.
  Divisor (2nd operand) is in S and T.



The Divide, DR, instruction divides the contents of B (high-order bits of dividend) and S (low-order bits of dividend) by the contents of T (divisor).

The dividend is a 64-bit signed integer occupying an even/odd pair of GPR's addressed by R1 and R1 + 1, respectively. To correctly specify the even/odd pair of GPR's, R1 must refer to an even-numbered GPR or a program specification interruption occurs. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered GPR, respectively, of LS. The divisor is also a 32-bit integer.

Two bits of the low-order half of the dividend are first placed into the high-order half of the dividend. A multiple of the divisor is then selected and subtracted from the high-order half of the dividend to form a partial remainder. The resultant value determines the partial quotient (PQ) bit which is placed into F to accumulate the bits until a PQ byte is available. This PQ byte is transferred to S, which contains the low-order half of the dividend, and replaces those bits that have already been used in the operation. This action continues until a complete quotient and remainder are available, at which time they are stored into LS and an end-op cycle is taken.

The sign of the quotient is determined algebraically; four possible combinations of signs can occur:

1. + dividend, + divisor, + quotient, + remainder.
2. - dividend, - divisor, + quotient, - remainder.
3. + dividend, - divisor, - quotient, + remainder.
4. - dividend, + divisor, - quotient, - remainder.

Note that if the dividend and divisor signs are alike the quotient is positive; if unlike the quotient is negative. Note also that the sign of the remainder is the same as the sign of the dividend, except for a zero result, which is always positive.

When the relative magnitude of the dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a program fixed-point divide interruption occurs. When this happens, the instruction is suppressed, leaving the dividend unchanged in local storage.

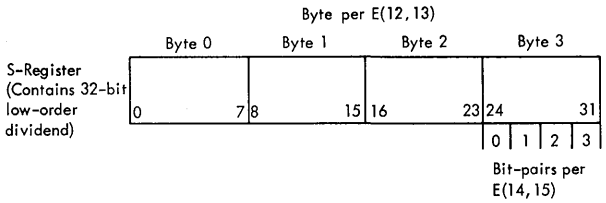
General Discussion

- Multiple selection bits are determined by E(12–15) and S bits.
- Multiples of divisor are determined by 'DVDL1' or 'DVDL0' micro-order, carry from PAL(28), and T(32).
- PQ bits are transferred and accumulated in F per E(14,15), and 'DVDL1' or 'DVDL0' micro-order.
- F contains byte of partial quotient when filled.
- Contents of F are transferred to correct byte in S per STC.

Execution of the DR instruction occurs in three iterative steps:

1. Transfer of bits from low-order half of dividend to high-order half of dividend.
2. Selection of divisor multiple.
3. Determination of quotient bits.

Selection of the two bits from the low-order half of the dividend is shown in B of Sheet 4, Diagram 5-110, FEMDM. Multiple selection bits (M1,M2) are selected from S per E(12–15), which is initially set to 0 and is incremented by 1 after the selection of each pair of multiple selection bits. E(12,13) determines which byte of S is selected, and E(14,15) determines which bit-pair of the selected byte will be used to set M1, M2:



M1,M2 has the same bit configuration as the bit-pair selected from S by E(12–15), and is inserted into B(64,65) [via PAL(64,65)] where it extends the high-order portion of the dividend in B.

Selection of the divisor multiple is shown in A of Sheet 4, Diagram 5-110. The factors which determine the divisor multiple are:

1. The carry condition from PAL(28) from the previous add cycle.
2. The state of T(32), which is the divisor sign bit.
3. The 'DVDL0' or 'DVDL1' micro-order.

Four divisor multiples are developed and applied as follows:

1. TL0 (+1 x T): The divisor in T is added to the partial remainder in AB.
2. TCL0 (-1 x T): The 2's complement of the divisor in T is added to the partial remainder in AB.
3. TL1 (+2 x T): The divisor in T is shifted left one bit position (in effect, doubled in value) and is added to the dividend or partial remainder in AB.
4. TCL1 (-2 x T): The 2's complement of the divisor in T is shifted left one bit position and is added to the dividend or partial remainder in AB.

The first divisor multiple is arbitrarily set to +2 x T if the divisor is negative (STAT G set) or to -2 x T if the divisor is positive (STAT G reset). This selection is done for two reasons:

1. The carry condition from PAL(28), which is normally a factor in selecting a divisor multiple, is meaningless at this time, because no previous reduction cycle has taken place.
2. If the relative magnitude of the dividend and divisor allows, the first reduction must be an overdraw. (Except for one special case, if the first reduction cycle does not result in an overdraw, the quotient and remainder will be invalid.)

Determination of the PQ bit is shown in C of Sheet 4, Diagram 5-110. The result of adding the selected divisor multiple to the dividend or partial remainder is stored into AB as a new partial remainder. The PQ bit is decoded as the inverse of A(28), and is gated to F (via SAL) per E(14,15) and the 'DVDL0' or 'DVDL1' micro-order in effect at that time. When a PQ byte has been accumulated in F, it is gated to S per the STC, where it replaces the byte of S which has already been used. When four PQ bytes have been stored into S, the complete quotient is in S and the remainder is in B.

At this point, STAT G is tested. (Recall that STAT G was set to the sign of the divisor.) If set, the divisor is negative and is in 2's complement form. Because the first reduction cycle is an attempt to overdraw the dividend, the negative divisor is shifted left 1 bit position (in effect, doubled in value) to PAA(31–62). If STAT G is reset, the divisor is positive and must be 2's complemented and shifted left one bit position to reduce the dividend. During the addition, the sign of the divisor is propagated into PAL(24–31), and the result (remainder) is gated to AB(24–67). A(28) is then tested to determine the PQ bit. A remainder in true form [A(28) = 0] causes a 1-bit to be selected for the PQ bit; a 2's complement remainder [A(28) = 1] causes a 0-bit to be selected for the PQ bit.

Two micro-orders, 'DVDL0' and 'DVDL1', are alternately used in the divide algorithm. Each micro-order has two functions: (1) to determine the location of the PQ bit in SAL (and F) from the bit-pair selected by E(14,15), and (2) to determine the shifting of the divisor multiple to PAA. [The carry condition from PAL(28) and the state of T(32) determine whether the multiple will be in true or 2's complement form.] The 'DVDL0' micro-order causes the selected PQ bit to be placed into the odd SAL bit position

of the bit-pair selected by E(14,15), thus locating the PQ bit in the PQ byte being accumulated in F. This micro-order also determines that the divisor multiple will be gated to PAA(32–63) (no shift). The 'DVDL1' micro-order causes the selected PQ bit to be placed into the even SAL bit position, and also determines that the divisor multiple will be gated to PAA(31–62) (shifted left 1 bit position).

Detailed Discussion

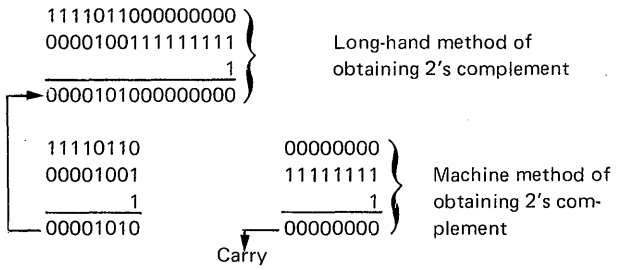
- Select M1,M2 bits from S per E(12–15).
- Insert M1,M2 bit-pair as low-order extension of high-order dividend in B.
- Select divisor multiple per: carry condition from PAL(28), divisor sign [T(32)], and 'DVDL0' or 'DVDL1' micro-order.
- Reduce dividend (or remainder) in AB by divisor multiple selected.
- Determine PQ bits per A(28).
- Accumulate PQ bits in F to form PQ byte.
- Accumulate PQ bytes in S to form quotient.
- Determine validity of quotient and remainder.

A flowchart of the DR instruction is shown in Diagram 5-110. At the start of execution, the instruction is in E, the high-order half of the dividend is in A, B, and D, and the second operand (divisor) is in S and T. (As previously mentioned, the dividend occupies an even-odd pair of GPR's.) To correctly specify the even-odd pair of GPR's, the R1 field of the instruction must refer to an even-numbered GPR, or a program specification interruption occurs.

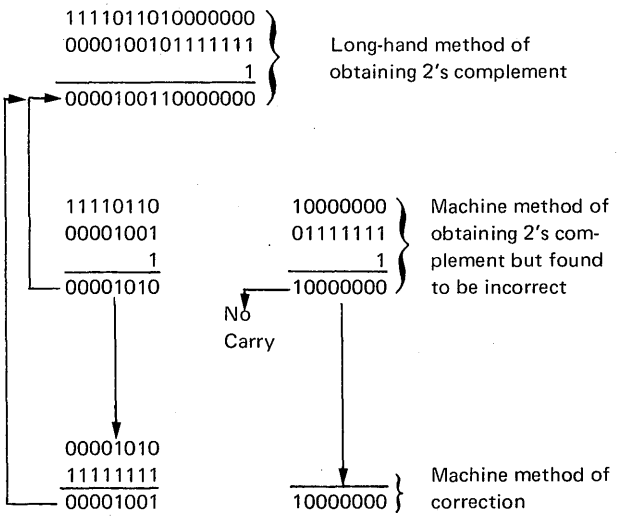
The states of B(32) and T(32) are first tested. B, at this time, contains the high-order half of the dividend, and T contains the divisor. If B(32) = 1, STAT B is set; if T(32) = 1, STAT G is set. These STAT's are used in later operations to determine the correct sign of the quotient and remainder, and to obtain a dividend in true form in B and S.

Recall that the dividend must be in true form. Because the dividend is a 64-bit operand and the maximum operable word length is 32 bits, the high-order half and the low-order half of the dividend must be treated separately when determining the true value of the dividend. To obtain a true value of a negative dividend, the dividend must be 2's complemented. Because the complementation process is always accomplished on a 2's complement basis during fixed-point operations, and because the high-order and low-order halves of the dividend are treated separately, the value of the high-order bits may be incorrect when in the true form. To prevent an incorrect high-order dividend from being operated on, the value of the low-order dividend bits is tested by checking for a carry out of the high-order bit location when 2's complemented. If a carry occurs, the high-order dividend bits are in the correct form. If a carry does not occur, the high-order bits are incorrect and a minus 1 must be added to the bits to obtain the desired dividend value. The following two examples illustrate this method (using only a 16-bit double word):

Example 1: Correct Value Obtained.



Example 2: Incorrect Value Obtained and How It Is Corrected.



Sheet 2 of Diagram 5-110 illustrates the method used by the CE to obtain a positive dividend value in B and S.

With the correct values of the dividend in B and S and the divisor in T, E(12–15) is set to 0000 to allow selection of the first byte of the low-order half of the dividend. The first partial remainder is obtained by transferring the contents of B (high-order half of dividend) to PAB(32–63) and placing the divide multiple selection bits, M1,M2, into PAL(64,65). The divide multiple selection bits are determined by decoding E(12–15) and the S bits, as previously described.

After selection of the M1,M2 bits, E(12–15) is incremented by 1, setting up conditions for selection of the next M1,M2 bits. The resultant value of the addition of M1,M2 to the high-order half of the dividend is transferred to AB(24–67), from where it is shifted left 2 into the parallel adder, introducing the M1,M2 bits into the partial remainder.

Recall that STAT G was set to the sign value of the divisor. At this point, STAT G is tested (Sheet 3 of Diagram 5-110.) If set, the divisor is negative and is in 2's complement form, and is transferred to the parallel adder, shifting left 1 to PAA(31–62). If STAT G is reset, the divisor is positive and must be 2's complemented before being transferred to the parallel adder so that it can reduce the dividend. During the addition, the sign of the divisor is propagated into PAL(24–31) and is checked after the addition to determine the PQ-bit setting.

At this point in the operation, the 'DVDL0' micro-order is in effect and causes the first PQ bit to be gated to the odd SAL bit position of the high-order bit-pair of SAL [E(14,15) = 01] (C of Sheet 4, Diagram 5-110). The first PQ bit is extraneous, because it reflects the condition of A(28) before the first reduction cycle, and it is replaced by a valid bit 2 cycles later. The result of the first reduction is placed into AB.

A 'DVDL1' micro-order is issued next. This micro-order causes selection of a PQ bit per A(28). (AB presently contains the result of the first reduction.) If the result of the first reduction is negative [A(28) = 1], a 0 is the selected PQ bit; if positive [A(28) = 0], a 1 is the selected PQ bit. Because the 'DVDL1' micro-order is now in effect, the PQ bit is gated to the even SAL bit position of the high-order bit-pair of SAL [E(14,15) = 01], and SAL is gated to F. The contents of AB are transferred to PAB, and the divisor multiple [selected by the carry condition of PAL(28), the state of T(32), and the 'DVDL0' micro-order] is gated to PAA. The result of the addition, together with a new pair of multiple selection bits, is gated to AB as a new partial remainder. The next divisor multiple is selected per the carry condition of PAL(28), the status of T(32) (which remains the same for the entire divide operation), and the 'DVDL1' micro-order. Refer to Table 3-3 for the value of the divisor multiple for all conditions.

Selection of the PQ bits and divisor multiples occurs as just described until a complete byte of the PQ is available: (1) the 'DVDL0' micro-order is issued, and the PQ bit obtained from the last addition is stored into the odd register location determined by E(14,15), (2) the next divisor multiple is selected, placed in the adder, and added to the partial remainder, and (3) a 'DVDL1' micro-order is issued. The 'DVDL1' micro-order accomplishes the same operation as the 'DVDL0' micro-order, except that it places the PQ bit into the even GPR location per E(14,15) and shifts the selected multiple left 1. (Refer to Sheet 3, Diagram 5-110.)

When a byte of the PQ is obtained, it is transferred to S per the STC, replacing those bits of the low-order half of the dividend that have already been used. Operations continue in the same manner; i.e., PQ bits are selected to form a PQ byte, this byte is transferred to S, and the used dividend bits are replaced until a complete quotient is developed.

Thirty-two reduction cycles are provided by the fixed-point divide microprogram. To obtain a valid quotient and remainder, the absolute value of the dividend and divisor must be so related that in 32 reduction cycles the dividend can be reduced to a remainder whose value is less than that of the divisor. If the highest-order significant bit of the dividend is less than 31 bit positions to the left of the highest-order significant bit of the divisor, the quotient is in true form and a valid result is obtained. If the highest-order significant bit of the dividend is more than 30 bit positions to the left of the highest-order significant bit of the divisor, the quotient is in 2's complement form and an invalid result is obtained unless the quotient is the maximum negative number (10000 . . . . . 000). Valid results are stored, but invalid results cause a program fixed-point divide interruption and the original operands remain unchanged in storage.

Table 3-3. Divide Multiple Values, Fixed-Point

Carry from PAL(28)		T(32)		Divide Multiple Micro-Order	
Yes	No	1	0	'DVDL0'	'DVDL1'
X			X	2's complement of T.	2's complement of T shifted left 1.
X		X		T.	T shifted left 1.
	X	X		2's complement of T.	2's complement of T shifted left 1.
	X		X	T.	T shifted left 1.

When E(14,15) = 11 and the STC = 11 (when tested), E(12–15) is incremented to 0000, the next-to-last PQ bit is stored into F(6), and the last reduction cycle is taken (Sheet 3, Diagram 5-110). The last PQ bit is taken from the remainder in AB [A(28)] and is gated to SAL(7) [per E(14,15) = 00 and the 'DVDL0' micro-order]. This action completes the last byte in SAL, which is gated to S per the STC.

A branch on divisor sign (STAT G) and quotient form (true or 2's complement, STAT C,) causes the microprogram to enter 1 of 4 termination routines. The purpose of the termination routines is to test for valid results, to establish the quotient and remainder in the proper form according to the proper sign, to store the corrected remainder and quotient, and to end the operation.

Sheet 6 of Diagram 5-110 is a flowchart of the fixed-point divide termination routine if the quotient is in true form. In this case, the quotient and remainder are valid, and it is only necessary to store the results in the proper form. (Positive results must be stored in true form and negative results must be stored in 2's complement form, according to the convention of fixed-point arithmetic and the rules of algebra.) Because the dividend was stored in B and S in true form regardless of sign, the results may or may not be in the correct form. It is therefore necessary to branch on the original signs of the operands and on the form of the results to achieve the proper form for the results. If the remainder is in 2's complement form, it is the result of an overdraw, and must be corrected by adding the divisor in true form before storing or complementing the remainder according to the sign of the dividend.

Sheet 5 of Diagram 5-110 is a flowchart of the fixed-point divide termination routine if the quotient is in 2's complement form. If the quotient is in 2's complement form and it is the maximum negative number (10000 . . . . . 000), the results are valid and the termination routine must accomplish the same tasks as for the true form termination. If the quotient is not the maximum negative number, a program fixed-point divide interruption is taken and the original operands remain unchanged in storage.

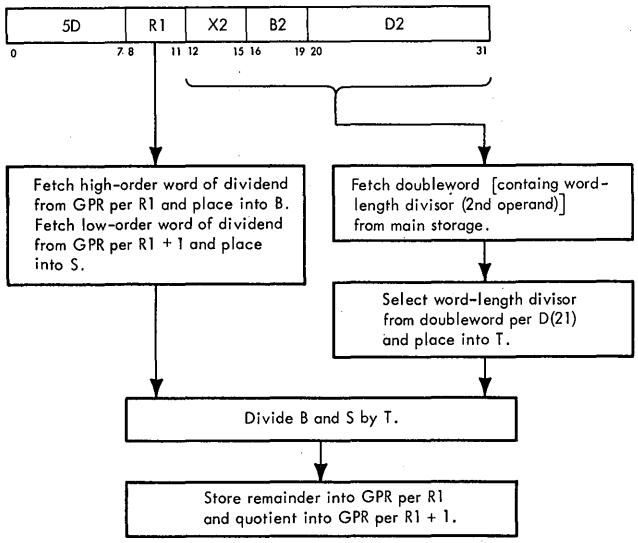
When the quotient and remainder are valid and have been converted to the proper form according to their algebraic sign, the remainder is stored into the GPR per E(8–11) and the quotient is stored into the GPR per E(8–11) + 1, replacing the high-order and low-order halves of the dividend respectively, and leaving the divisor unchanged in storage. An end-op cycle is taken, finishing the operation.

Two examples of the fixed-point divide operation are presented in Figures 3-8 and 3-9. These examples are the inverse of the two fixed-point multiply examples (Figures 3-6 and 3-7).

Divide, D (5D)

- Divide 1st operand (in GPR per R1 and R1 + 1) by 2nd operand (in storage) and place result into 1st operand location (remainder in GPR per R1; quotient in GPR per R1 + 1).

- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  High-order half of dividend (1st operand) is in S and T.
  Low-order half of dividend, is in GPR per R1 + 1.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.

- D(21) determines which word of doubleword contains divisor: if a 1, right word; if a 0, left word.

The Divide, D, instruction divides the contents of B (high-order bits of dividend) and S (low-order bits of dividend) by the contents of T (divisor). D(21) determines which word of the doubleword fetched from main storage contains the divisor: if a 1, right word; if a 0, left word. Once the divisor is obtained from main storage, the operation is identical to the operation of the DR instruction (Diagram 5-110).

CONVERT

There are two fixed-point convert instructions, both in the RX format: Convert to Binary and Convert to Decimal. These instructions convert the radix of an operand from decimal to binary and binary to decimal, respectively, and

	Hex	Binary	Decimal
Dividend (in B and S)	108E	0001 0000 1000 1110	+4238
Divisor (in T)	001A	0000 0000 0001 1010	+26
Quotient (to S via F)	00A3	0000 0000 1010 0011	+163

1. M1, M2 bits are derived from S per E(12-15).
2. Multiple selection per PAL(28) carry, T(32), and 'DVDL0' and 'DVDL1' micro-orders (Table 3-3).
3. T = T-register.

	Accumulate PQ Bits in F, Byte to S	A	B	B(64,65)	Remarks	M1, M2 (S Bits)	Carry A(28)	T(32)	Micro-order
			0000 0000 0001	00	Dividend + M1, M2 to AB	00		0	
		00	0000 0000 0100		AB left 2			0	
		1111	1111 1100 1100		+ T (2's complement) left 1		-	0	DVDL1
		1111	1111 1101 0000		Remainder to AB			0	
	0000 0	0000	0000 0001 1010		+ T		0	0	DVDL0
		1111	1111 1110 1010	00	Remainder to AB	00		0	
	0000 00	1111	1111 1010 1000		AB left 2			0	
		0000	0000 0011 0100		+ T left 1		0	0	DVDL1
		1111	1111 1101 1100		Remainder to AB			0	
	0000 000	0000	0000 0001 1010		+ T		0	0	DVDL0
		1111	1111 1111 0110	10	Remainder to AB	10		0	
To S	0000 0000	1111	1111 1101 1010		AB left 2			0	
		0000	0000 0011 0100		+ T left 1		0	0	DVDL1
		0000	0000 0000 1110		Remainder to AB			0	
	1	1111	1111 1110 0110		+ T (2's complement)		1	0	DVDL0
		1111	1111 1111 0100	00	Remainder to AB	00		0	
	10	1111	1111 1101 0000		AB left 2			0	
		0000	0000 0011 0100		+ T left 1		0	0	DVDL1
		0000	0000 0000 0100		Remainder to AB			0	
	101	1111	1111 1110 0110		+ T (2's complement)		1	0	DVDL0
		1111	1111 1110 1010	11	Remainder to AB	11		0	
	1010	1111	1111 1010 1011		AB left 2			0	
		0000	0000 0011 0100		+ T left 1		0	0	DVDL1
		1111	1111 1101 1111		Remainder to AB			0	
	1010 0	0000	0000 0001 1010		+ T		0	0	DVDL0
		1111	1111 1111 1001	10	Remainder to AB	10		0	
	1010 00	1111	1111 1110 0110		AB left 2			0	
		0000	0000 0011 0100		+ T left 1		0	0	DVDL1
		0000	0000 0001 1010		Remainder to AB			0	
	1010 001	1111	1111 1110 0110		+ T (2's complement)		1	0	DVDL0
		0000	0000 0000 0000		Remainder to AB			0	
To S	1010 0011	S ⟶ 0000 0000 1010 0011 = 00A3 Hex							

Figure 3-8. Fixed-Point Divide, Example No. 1

	Hex	Binary	Decimal
Dividend (in B and S)†	108E	0001 0000 1000 1110	-4238
Divisor (in T)	FFE6	1111 1111 1110 0110	-26
Quotient (to S via F)	00A3	0000 0000 1010 0011	+163

† Dividend is always in true form in B and S.

1. M1, M2 bits are derived from S per E(12-15).
2. Multiple selection per PAL(28) carry, T(32), and 'DVDL0' and 'DVDL1' micro-orders (Table 3-3).
3. T = T-register.

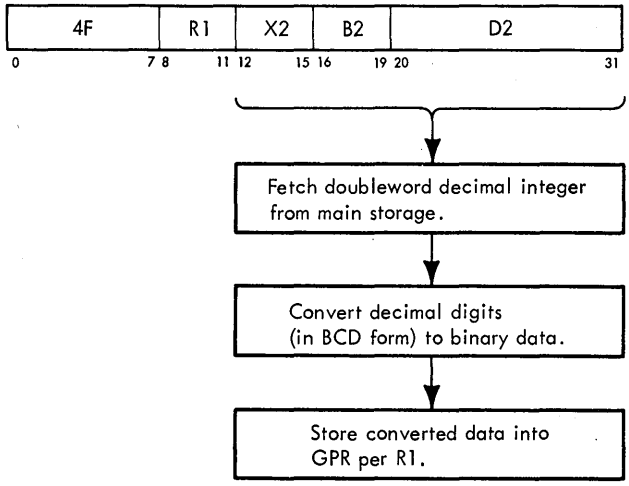
	Accumulate PQ Bits in F, Byte to S	A	B	B(64,65)	Remarks	M1, M2 (S Bits)	Carry A(28)	T(32)	Micro-order
			0000 0000 0001	00	Dividend + M1, M2 to AB	00		1	
		00	0000 0000 0100		AB left 2			1	
		1111	1111 1100 1100		+ T left 1		-	1	DVDL1
		1111	1111 1101 0000		Remainder to AB			1	
	0000 0	0000	0000 0001 1010		+ T (2's complement)		0	1	DVDL0
		1111	1111 1110 1010	00	Remainder to AB	00		1	
	0000 00	1111	1111 1010 1000		AB left 2			1	
		0000	0000 0011 0100		+ T (2's complement) left 1		0	1	DVDL1
		1111	1111 1101 1100		Remainder to AB			1	
	0000 000	0000	0000 0001 1010		+ T (2's complement)		0	1	DVDL0
		1111	1111 1111 0110	10	Remainder to AB	10		1	
To S	0000 0000	1111	1111 1101 1010		AB left 2			1	
		0000	0000 0011 0100		+ T (2's complement) left 1		0	1	DVDL1
		0000	0000 0000 1110		Remainder to AB			1	
	1	1111	1111 1110 0110		+ T		1	1	DVDL0
		1111	1111 1111 0100	00	Remainder to AB	00		1	
	10	1111	1111 1101 0000		AB left 2			1	
		0000	0000 0011 0100		+ T (2's complement) left 1		0	1	DVDL1
		0000	0000 0000 0100		Remainder to AB			1	
	101	1111	1111 1110 0110		+ T		1	1	DVDL0
		1111	1111 1110 1010	11	Remainder to AB	11		1	
	1010	1111	1111 1010 1011		AB left 2			1	
		0000	0000 0011 0100		+ T (2's complement) left 1		0	1	DVDL1
		1111	1111 1101 1111		Remainder to AB			1	
	1010 0	0000	0000 0001 1010		+ T (2's complement)		0	1	DVDL0
		1111	1111 1111 1001	10	Remainder to AB	10		1	
	1010 00	1111	1111 1110 0110		AB left 2			1	
		0000	0000 0011 0100		+ T (2's complement) left 1		0	1	DVDL1
		0000	0000 0001 1010		Remainder to AB			1	
	1010 001	1111	1111 1110 0110		+ T		1	1	DVDL0
		0000	0000 0000 0000		Remainder to AB			1	
To S	1010 0011	S ⟶ 0000 0000 1010 0011 = 00A3 Hex							

Figure 3-9. Fixed-Point Divide, Example No. 2

store the result into the other operand location. The maximum positive number that can be converted is +2,147,483,647; the maximum negative number is -2,147,483,648.

Convert to Binary, CVB (4F)

- Convert radix of 2nd operand (doubleword, in storage) from decimal to binary and place result into 1st operand location (in GPR per R1).
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- Decimal digits are in packed format.
- High-order half of doubleword operand is converted first.
- For numbers outside maximum range, only 32 low-order binary bits are stored after conversion.
- Conversion is accomplished by multiplying decimal data by 10, adding result to next decimal digit, then multiplying this total by 10, and so on.

The Convert to Binary (CVB) instruction converts the radix of the second operand from decimal to binary, placing the result into the first operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The decimal operand (second operand) occupies a doubleword in main storage and has a packed decimal data format. The low-order four bits of the operand represent the sign of the operand. The remaining 60 bits contain 15 binary-coded decimal (BCD) digits in true notation. The decimal data, when being converted, is tested for valid sign and digit codes. If improper codes exist, a program data interruption occurs.

The result of the conversion is placed in the GPR specified by R1. The maximum positive number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the maximum negative number is -2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 low-order binary bits into the register, and initiating a program fixed-point divide check interruption. In the case of a negative second operand, the low-order part is in 2's complement notation.

Converting a decimal number to binary involves taking the BCD digits of the number to be converted one at a time, from the most significant to the least significant (left to right). The leftmost digit is taken from the decimal field (in BCD form) and multiplied by 10. The next most-significant digit is taken from the decimal field and added to this result. The result of this addition is then multiplied by 10. The operation of adding and multiplying by 10 is repeated until all digits have been converted; however, no multiplication takes place after the least-significant decimal digit has been added.

The following is an example of how the computer basically accomplishes the conversion:

Convert 146 BCD to Binary

		BCD	146 = 0001	0100	0110
1.	1st BCD digit to PAA	(x1)	0001		
2.	1st BCD digit left 2 (times 4 to PAB	(x4)	000100		
3.	Add; result is 1st BCD digit times 5	(x5)	000101		
4.	Result of step 3 left 1 (times 2) to PAA	(x10)	0001010		
5.	2nd BCD digit to PAB	(x1)	0100		
6.	Add; result is 1st BCD digit times 10, plus 2nd BCD digit		0001110		
7.	Result of step 6 to PAA	(x1)	0001110		
8.	Result of step 6 left 2 (times 4) to PAB	(x4)	000111000		
9.	Add; result is 1st BCD digit times 50, plus 2nd BCD digit times 5	(x5)	001000110		
10.	Result of step 9 left 1 (times 2) to PAA	(x10)	0010001100		
11.	3rd BCD digit to PAA	(x1)	0110		
12.	Add; result is 1st BCD digit times 100, plus 2nd BCD digit times 10, plus 3rd BCD digit		0010010010 = 146 Binary		

Diagram 5-111, FEMDM, is a flowchart of the CVB instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T (not used), the second operand address is in D, and a storage request for the second operand has been issued per D. At the beginning of the operation, the address of the second operand is tested to see whether the proper integral boundary has been specified. If the address is located on an incorrect (non-doubleword) integral boundary, a program specification interruption occurs. The CVB operation is suppressed, and the data in LS and in main storage remains unchanged. If no program specification interruption occurs, the operation continues.

Recall that, at the start of execution, a storage request for the second operand had been issued. At this time, the data (doubleword operand) is present at the SDBO and is gated into ST, destroying the first operand. The STC, which selects the correct byte to be converted from ST, is set to 000, thus selecting S(0–7).

The contents of T (low-order half of the doubleword operand) are transferred to the LSWR. (This data is converted at a later time.) Because the converted data is to be stored into T and D, they are now cleared. The first byte (bits 0–7) of S is transferred to the serial adder B bus. Bits 0–3 are transferred to SAL(0–3) and SAL(4–7) and on to F. F now contains the first decimal digit to be converted.

As the decimal data from S is passed through the serial adder to F, it is tested for invalid digits. If the digits are invalid, STAT E is set and later, when tested, causes a program data interruption, which terminates the operation. If the digits are valid, the first decimal digit is transferred from F(4–7) to PAB(60–63). The contents of D and T are then transferred and shifted left 1 to PAA(7–30) and PAA(31–62), respectively, and added to the decimal digit. At this time, D and T contain zero.

The result of the addition, which is the decimal digit, is transferred from PAL(8–63) to DT and from PAL(32–63) to B(32–63). The contents of DT are then transferred to PAA(8–63). The contents of B are now shifted left 2, placed into PAB(4–65), and added to PAA. This action, in effect, multiplies the original decimal digit by 5. The result of the addition is then transferred from PAL(8–63) to DT. A byte from S is transferred to the serial adder B bus per the STC. At this time, the byte transferred to the serial adder is the first byte in S (STC = 000). This action allows the second decimal digit to be placed into the serial adder. From the serial adder, the data is sent to F. The STC is increased by 1 so that the next byte from S can be transferred when selected.

The second decimal digit, F(4–7), is transferred to PAB(60–63). The contents of DT are then transferred to PAA(7–62). This transfer shifts the converted data left 1, in effect multiplying the original decimal digit by 10 (x5 and x2). PAA and PAB are added, and the result in PAL(8–63) is transferred to DT. The contents of PAL(32–63) are transferred to B(32–63).

The next byte in S (bits 8–15) is now transferred to F via SAL. STAT D is then tested. If STAT D is set, it indicates the low-order word of the doubleword operand is being converted; if reset, the high-order word is being converted. At this time, STAT D is reset. The STC is now tested to see which byte of S is being worked on; the value presently in the STC is 001. Because STAT D is reset and the STC does not equal 011, operations continue in the same manner as previously described; i.e., a decimal digit is brought in and added to the sum of the converted digits, and the result is multiplied by 10. This procedure continues until all digits have been converted.

While the last two decimal digits of the high-order word are being processed, STAT D is set to indicate the low-order word. When the last decimal digit of the high-order word has been transferred to F, the STC is set to 000; the low-order word is transferred from the LSWR (where it was stored at the beginning of the operation) to S and is converted in the same manner as the high-order word.

When the low-order byte of the low-order word is transferred from SAL to F, the sign is tested for validity, setting STAT E if invalid. The state of STAT E is then tested. If STAT E is set, a data-check condition exists and an end-op cycle is taken, leaving the contents of LS unaltered. If STAT E is not set, the sign of the number is determined by [F(4–7)], and the last decimal digit is converted. If F(4–7) is a plus sign, the converted data is transferred from T to the GPR per E(8–11). If F(4–7) is a minus sign, the converted data is first 2's complemented and then transferred from T to the GPR per E(8–11). STAT G is then set if T(32) = 1. PAL(32–63) is tested for all zero's (T = 0), and a branch is made on the result of this test.

The contents of D (overflow bits) are then transferred to PAL(40–63), and PAL(32–63) is again tested for all zero's (D = 0). Note that D is not 2's complemented when T is 2's complemented for a negative sign, and should always equal zero unless an overflow occurred.

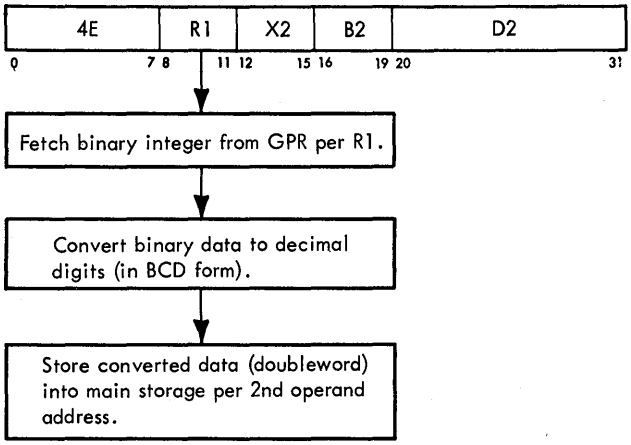
If T(32–63) = 0 and D(0–23) = 0, the result is zero and a normal end-op cycle is taken. In all cases, if D(0–23) does not equal 0, an overflow has occurred, a fixed-point divide check condition exists and an end-op cycle is taken. If D(0–23) = 0 and T(32–63) does not equal 0, a further test is made to determine if the maximum positive or negative number has been exceeded. If the decimal sign [F(4–7)] was positive and T(32) = 1 (STAT G set), the maximum positive number has been exceeded. If the decimal sign [F(4–7)] was negative and T(32) = 0 (STAT G reset), the maximum negative number has been exceeded. (For a negative sign, the contents of T have previously been changed to 2's complement form.) In both of the above cases, a fixed-point divide check condition exists and an

end-op cycle is taken. If the maximum number has not been exceeded, an end-op cycle is taken, completing the operation.

Note that even if an overflow condition is detected or if the maximum positive or negative number has been exceeded, the low-order 32 bits of the converted integer are stored into the GPR per E(8–11) and the only indication is the fixed-point divide check condition. If any decimal digit or the sign is invalid, a data check condition exists and the operation is terminated without storing any data.

Convert to Decimal, CVD (4E)

- Convert radix of 1st operand (in GPR per R1) from binary to decimal and place result into 2nd operand location (in storage).
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- Operand to be converted is 32-bit signed binary integer.
- Converted data is in packed decimal format.
- Positive sign is encoded as 1100 or 1010.
- Minus sign is encoded as 1101 or 1011.

The Convert to Decimal (CVD) instruction converts the radix of the first operand (from GPR per R1) from binary to decimal, storing the result into the second operand location (in main storage). The number to be converted is a 32-bit signed binary integer; the 31 binary bits yield 15 decimal digits in the packed format. The sign bit may be encoded in two forms for both positive and negative numbers. A positive sign may be 1100 or 1010; a minus sign may be 1101 or 1011. The choice between the two sign representations is determined by PSW(12).

A binary number is converted to decimal by parallel decimal correct function, which operates from AB to the parallel adder. The binary number being converted is extracted one bit at a time from the most significant bit to the least significant bit (left to right) and is added to a previous partially converted decimal number. The resultant number is first multiplied by 2, by shifting left 1, and then decimal-corrected. The parallel decimal-correct function extracts 6 or 0 from the partially converted number and adds the 6 or 0 to twice the partially converted number so that the resulting decimal number does not exceed the maximum decimal number of 9. This function is performed for each half-byte of AB(28–63) for every add cycle. Table 3-4 lists the AB bits and parallel adder bits used in the decimal-correct function. The process of conversion is repeated until all bits of the binary number have been examined. At the completion of the conversion, the converted number is shifted left 4, and the correct sign is placed into the low-order bit positions. The example shown in Figure 3-10 illustrates the method of converting from binary to decimal.

Table 3-4. Conversion to Decimal (Excess-6)

AB Bits Set	Set PAB Bus Bits (+6)
A(28)	PAB(29,30)
A(29,30)	PAB(29,30)
A(29) and A(31)	PAB(29,30)
B(32)	PAB(33,34)
B(33,34)	PAB(33,34)
B(33) and B(35)	PAB(33,34)
B(36)	PAB(37,38)
B(37,38)	PAB(37,38)
B(37) and B(39)	PAB(37,38)
B(40)	PAB(41,42)
B(41,42)	PAB(41,42)
B(41) and B(43)	PAB(41,42)
B(44)	PAB(45,46)
B(45,46)	PAB(45,46)
B(45) and B(47)	PAB(45,46)
B(48)	PAB(49,50)
B(49,50)	PAB(49,50)
B(49) and B(51)	PAB(49,50)
B(52)	PAB(53,54)
B(53,54)	PAB(53,54)
B(53) and B(55)	PAB(53,54)
B(56)	PAB(57,58)
B(57,58)	PAB(57,58)
B(57) and B(59)	PAB(57,58)
B(60)	PAB(61,62)
B(61,62)	PAB(61,62)
B(61) and B(63)	PAB(61,62)

Diagram 5-112, FEMDM, is a flowchart of the CVD instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the second operand address is in D. The first portion of the

Convert +146 Binary (92 Hex) to +146 BCD

1. SAL = 1001 0010 (shift left 1 after each addition).
2. Gate 6 or 0 to each half-byte of PAB per AB bits (see Table 3-4).

AB												Remarks
52	53	54	55	56	57	58	59	60	61	62	63	
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	0	0	0	0	0	0	1	Sum to AB and DT.
0	0	0	0	0	0	0	0	0	0	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
0	0	0	0	0	0	0	0	0	0	1	0	Sum to AB and DT.
0	0	0	0	0	0	0	0	0	1	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
0	0	0	0	0	0	0	0	0	1	0	0	Sum to AB and DT.
0	0	0	0	0	0	0	0	1	0	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	0	0	0	1	0	0	1	Sum to AB and DT.
0	0	0	0	0	0	0	1	0	0	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	0	0	0	0	1	1	0	0	0	Sum to AB and DT.
0	0	0	0	0	0	1	1	0	0	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	0	0	0	1	1	0	1	1	0	Sum to AB and DT.
0	0	0	0	0	1	1	0	1	1	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	1	1	1	0	0	1	1	Sum to AB and DT.
0	0	0	0	1	1	1	0	0	1	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	1	1	0	0	0	0	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	1	0	1	0	0	0	1	1	0	Result [place sign of 1100 into B(64-67) per STAT C].

0001 0100 0110 1100 = +146 in BCD packed format

Figure 3-10. Convert to Decimal Example

operation is devoted to testing for specification-check and address-store-compare conditions. If a specification check is present, a program specification interruption occurs and the operation is suppressed. If an address-store-compare condition occurs, the 'PSC' trigger is set and the operation continues. The value of S(0) (sign of original binary number) is set into STAT C. At the end of the operation, STAT C is examined to determine the sign of the converted number.

Because conversion is done on a positive operand basis, the sign of the operand is determined by testing T(32). If T(32) is a 1, the data to be converted is negative and its 2's complement form must be derived; if a 0, the data is positive. The contents of D are then shifted left 4 and transferred to the LSWR. Because the high-order converted data is stored in D, D is cleared. The first byte of data is now sent from S to SAL per the STC (STC = 0).

The first decimal convert value (0 or 6) is obtained by examining the contents of AB. Because AB is cleared at this time, the first decimal convert value is all 0's. (See Table 3-4 for conversion values.) The decimal convert value is placed into PAB(28–63), and a hot carry is generated if SAL(0) = 1. If SAL(0) = 0, a hot carry is not generated and the value in PAB is transferred directly to PAL. The contents in SAL are shifted left 1, thus placing the next bit from the byte into SAL(0). Next, PAL(8–63) is transferred to AB(8–63), PAL(8–31) is transferred to D(0–23), and PAL(32–63) is transferred to T(32–63). This action places the first converted bit into AB and DT. The value in DT is then shifted left 1 to PAA(7–63). A decimal convert value is again obtained from B and placed into PAB. SAL(0) is now checked. If it is a 1, a hot carry is generated; if a 0, no carry is generated. In either case, the numbers are added. The result of the addition is then transferred to DT and AB.

The contents of SAL are shifted left 1, bringing in the next bit for conversion. The ABC is increased by 1.

The contents of SAL (a byte of S per the STC) are shifted left one digit position as follows: F is gated to SAA, the byte of S (per the STC) is gated to SAB, and SAL is gated to F and S (per the STC). This operation is equivalent to adding the contents of SAL to itself, which doubles the value of SAL and, in effect, shifts the bits one digit position to the left. After seven add cycles, every bit in SAL has been presented in sequence to PAA(63) via SAL(0), the original low-order bit of F (and of the byte of S) has been shifted to the high-order bit position of F and of the byte of S, and the STC has been incremented by 1 to present the next byte of S to SAL and F. To assure that the new byte of S presented to SAB will be added to all zeros, F(4–7) is gated to SAA(0–3) and all zeros are gated to SAA(4–7).

The ABC indicates the progress of the microprogram in converting bits of the byte from S (now located in SAL, F, and S). When the ABC = 4, five bits of the byte have been converted, the microprogram has branched (per ABC = 3) to a routine to convert the last three bits of the byte, the ABC is set to 0, and the STC is incremented by 1 to present the next byte of S to SAL and F. One conversion cycle is taken and the microprogram re-enters the conversion loop, converting the next byte of S in the same manner as the first. This routine continues until, at the time the microprogram branches from the conversion loop, the STC = 3. The microprogram then enters a termination routine, during which the last three bits of the low-order byte of S are converted and the sign is set per STAT C. (Recall that STAT C was set to the value of the sign of the binary number.) If STAT C is set, the last decimal convert value (converting bit 7 of the last byte) is obtained and the hot carry is generated according to the value of SAL(0); the result is placed into AB and DT. A minus sign (1101 or 1011) is then placed into B(64–67). AB is shifted left 4 and transferred to ST(0–63). Mark triggers 0 through 7 are set, and the converted data is transferred to main storage. An end-op cycle is taken, completing the operation. If STAT C is reset, a positive sign is placed into B(64–67), shifted into the low-order bits of the converted operand, and stored into main storage.

The CVD instruction is the only instruction which uses the excess-6 gates from B to PAB. (For a discussion of the excess-6 gates, refer to "Parallel Adder" in Chapter 2.) Note in the example in Figure 3-10 that the result of each addition is gated from PAL to AB and to DT; then the contents of DT are gated left 1 to PAA (in effect, doubled in value). By examining the bit configuration of each half-byte of AB, a decision is made whether to add 6 to the corresponding half-byte of DT after being shifted left 1 to PAA. This decision is a prediction whether or not each half-byte of AB, if doubled in value, will exceed 9. Each half-byte of AB which has a value of 5 or greater will, if doubled, exceed 9. Therefore, for decimal correction purposes, 6 must be added to the corresponding half-bytes of DT.

For simplicity, the low-order half-byte of B is taken as an example in Table 3-5; however, each half-byte of AB (28–63) is simultaneously examined. Note in the table that if B(60–63) contains a value of 0 through 4, the value, when doubled, will not exceed 9 and a 6 is not added. If B(60–63) contains a value of 5 through 15, however, its value, when doubled, will exceed 9 and a 6 must be added. Note that for values 5 through 15 B(60–63) contains one or more of the following bit combinations: B(60) = 1, B(61,62) = 11, B(61) and B(63) = 1.

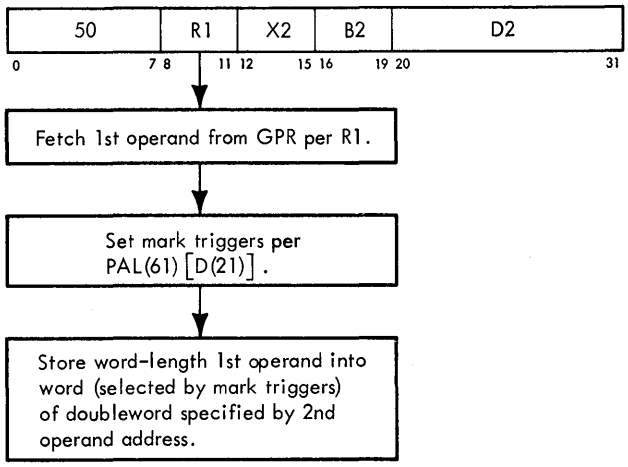
STORE

There are three fixed-point store instructions: Store, ST, RX format; Store Halfword, STH, RX format; and Store Multiple, STM, RS format. The function of the store instructions is to store the contents of specified GPR(s) into main storage.

Store, ST (50)

- Store 1st operand (in GPR per R1) into 2nd operand location (in storage).

- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- PAL(61) determines into which word of doubleword 1st operand is to be stored: if a 1, right word; if a 0, left word.

The Store, ST, instruction stores the first operand (from the GPR per R1) into the main storage address specified by the second operand address. Diagram 5-113, FEMDM, is a

Table 3-5. Excess-6 Conversion, B(60–63)

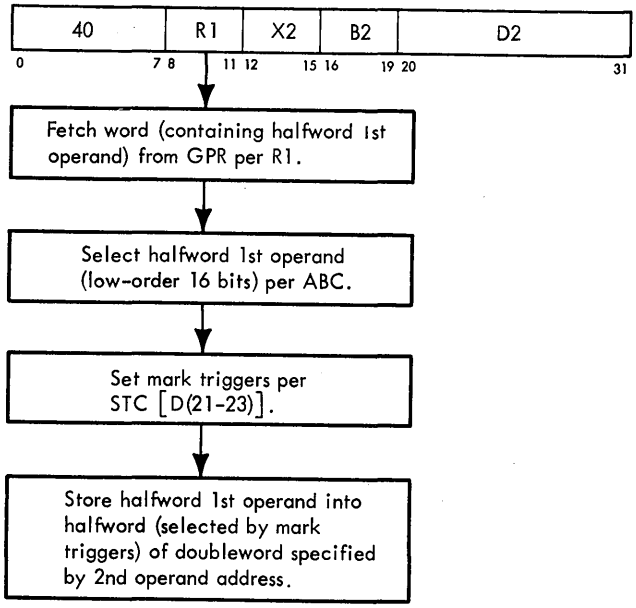
Decision Making Factors (ALD RB 753)	B-bits 60	61	62	63	Decimal Value	Decimal Value if Doubled	Add to PAB(60–63)
	0	0	0	0	0	0	
None of the	0	0	0	1	1	2	
below conditions	0	0	1	0	2	4	0
	0	0	1	1	3	6	
	0	1	0	0	4	8	
B(61) and B(63) = 1	0	1	0	1	5	10	
B(61,62) = 11	0	1	1	0	6	12	
	0	1	1	1	7	14	
	1	0	0	0	8	16	
	1	0	0	1	9	18	6
	1	0	1	0	10	20	[1 to
B(60) = 1	1	0	1	1	11	22	PAB(61,62)]
	1	1	0	0	12	24	
	1	1	0	1	13	26	
	1	1	1	0	14	28	
	1	1	1	1	15	30	

Refer to Table 3-4 for remaining half-bytes of AB.

flowchart of the ST instruction. PAL(61) determines which word (right or left) of the doubleword addressed by D will receive the first operand; mark triggers 0–3 or 4–7 are set accordingly.

Store Halfword, STH (40)

- Store halfword 1st operand (in GPR per R1) into 2nd operand location (in storage).
- RX format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- ABC selects 16 low-order bits of 1st operand for storage; high-order bits are ignored.
- STC [D(21–23)] positions 16 low-order bytes for storage.

The Store Halfword (STH) instruction stores the 16 low-order bits of the GPR specified by R1 into the main storage location specified by the second operand address. The high-order bits are not used.

The selected halfword is stored through the use of mark triggers, which reflect the value of D(21–23). This value, plus 1, signifies into which portion of the doubleword the halfword is to be stored. Diagram 5-114, FEMDM, is a flowchart of the STH instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the second operand address (into which the halfword operand is to be stored) is in D.

The second operand address is first tested to see that it is on an integral boundary; if not, a program specification interruption occurs and the operation is suppressed. Assuming no specification error, D(21–23) is transferred to the STC to select the correct portion of the main storage doubleword into which the 16 low-order bits of the first operand are to be stored. Next, the first operand (located in T) is transferred to B. The ABC is then set to 6 by placing all 1's into the ABC and subtracting 1 from this value. (The ABC selects the two low-order bytes of the operand presently located in B.) The mark trigger, which transfers the selected high-order byte of the halfword from ST to the main storage location, is selected per the STC. (See Table 3-6 for the STC and mark trigger settings and the corresponding operand bits transferred.) The eight high-order bits of the halfword are now transferred from B(48–55) to the correct position via the serial adder and

Table 3-6. Operand Bits Transferred, STH Instruction

STC			Mark Trigger	Operand Bits Transferred
D(21)	D(22)	D(23)		
0	0	0	0	0–7
0	1	0	2	16–23
1	0	0	4	32–39
1	1	0	6	48–55
STC +1				
0	0	1	1	8–15
0	1	1	3	24–31
1	0	1	5	40–47
1	1	1	7	56–63

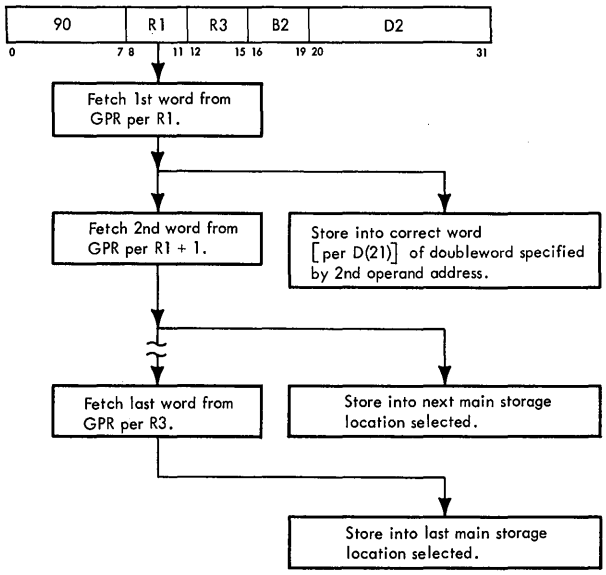
the STC. A 3-cycle storage request is given. (Three cycles later, the data in ST is stored into main storage.) Also at this time, the ABC and STC are increased by 1 to select the next byte of data and to position the byte into ST by the time the 3-cycle storage request has elapsed.

To determine whether this store operation modified the instruction to be executed next, an address-store-compare test is made by comparing the IC with the main storage address used in the store operation. The test is made by transferring the 2's complement of D (address of main storage doubleword into which the halfword operand is to be stored) to PAA(40–63). The contents of the IC are transferred to PAB(40–63); PAA and PAB are added, and the result is shifted right 4 in PAL. The PAL is tested for 0; if 0, an address-store-compare condition exists and the 'PSC' trigger is set. This trigger is tested during the I-Fetch sequence of the next instruction and, if set, causes the modified instruction to be fetched from main storage and reloaded in Q. (Refer to "ASC Test" in Section 1 of this Chapter for an explanation of the address-store-compare test during I-Fetch.)

At this point, the eight low-order bits of the halfword operand are transferred from B(56–63) to ST via the SAL and STC. The associated mark triggers are also set at this time per the STC. An end-op cycle is taken to complete the operation. If a protection check condition occurs, the operation is suppressed, because no data storage takes place. Instead, the next instruction is fetched and executed, followed by a program interruption due to the "late" protection check.

Store Multiple, STM (90)

- Store into 2nd operand location (as many words as required, in storage) contents of GPR's, in ascending order, starting with 1st operand location (per R1) and ending with 3rd operand location (per R3).
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- Number of words to be stored is determined by E(8–11) and E(12–15).
- Addressed GPR's wrap around from 15 to 0.
- D(21) determines into which word of doubleword the first word is to be stored: if a 1, right word; if a 0, left word.

The Store Multiple (STM) instruction stores one or more 32-bit words from LS, starting with the GPR specified by R1 and ending with the GPR specified by R3, into main storage. The area in main storage where the contents of the GPR's are placed starts at the location designated by the second operand address and continues through as many words as needed in an ascending order.

The number of words to be stored is determined by E(8–11) and E(12–15). If the contents of these bit locations are equal, only one word is to be stored. If the contents of the bit locations are not equal, storage of words is continued; E(8–11) is updated by 1 for each word stored until the bit locations are equal; then, one more word is stored, and the operation is completed. Once it has been decided whether one or more words are to be stored, it must be determined into which word of the doubleword, in main storage, the first word is to be stored; D(21) serves this function. If D(21) = 0, the first word is to be placed into the left word of the doubleword; if D(21) = 1, into the right word.

See Diagram 5-115, FEMDM, a flowchart of the STM instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in ST, and the second operand address is in D. The instruction first tests for a specification-check condition. If one exists, a program specification interruption occurs and the operation is suppressed. Assuming there is no specification check, the contents of D are transferred to PAA(40–63). A 3-cycle storage request is given. To determine whether one or more words are to be stored, E(8–11), R1, is compared with E(12–15), R3. If E(8–11) equals E(12–15), only one word is to be stored; if it does not, more than one word is to be stored.

Assume one word is to be stored [E(8–11) equals E(12–15)]. D(21) is tested to determine whether the LS operand is to be stored into the left or right word of the main storage doubleword. If D(21) = 0, the left word is selected; if D(21) = 1, the right word is selected. Assume D(21) = 1. Mark triggers 4 through 7 are set to gate T(32–63) to bits 32–63 of the main storage doubleword when data is stored. An address-store-compare test is then made. This test involves transferring the 2's complement of D to PAA(40–63) and 7 to PAA(61–63), and transferring the contents of the IC to PAB(40–63); they are added and shifted right 4 to PAL. The PAL is then tested for zero. If zero, the 'PSC' trigger is set. This trigger is tested during end op and, if set, indicates that the next instruction to be executed has been modified. The modified instruction must then be refetched into Q during I-Fetch. (For information about the address-store-compare test, refer to "ASC Test" in Section 1 of this chapter.)

A protection test is also made by main storage while the next instruction is being fetched. The protection key in the PSW is compared with the storage key for the location. If the keys agree, storage is permitted. If the keys do not agree, storage is not permitted, the instruction is terminated, and a "late" protection interruption is taken after the execution of the next instruction.

Now assume D(21) = 0 and E(8–11) equals E(12–15). Again only one word is to be stored into main storage. In this case, however, the word is to be stored in the left word of the doubleword location in main storage. Accordingly, mark triggers 0 through 3 are set to gate S(0–31) to bits 0–31 in main storage.

Now assume E(8–11) is not equal to E(12–15). If D(21) = 1, the first word is to be stored into the right word of the doubleword (mark triggers 4 through 7 are set). Because more than one word is to be loaded, the next sequentially addressed word from LS is transferred to S. A storage request is then given. The contents of D are increased by 8. E(8–11) and E(12–15) are again compared. If equal, mark triggers 0 through 3 are set and the data is gated into main storage. If they are not equal, more than two words are to be stored into main storage. Because the second operand to be gated from LS is already in S, and because D contains the address where both the second and third operands are to be stored, the third operand is now transferred out of LS and mark triggers 0 through 7 are set. The data is stored into main storage. Address-store-compare and protection-key tests are made for each new main storage address. Only the last storage request issued can cause a "late" protection interruption.

If more than two words are to be stored, the operation continues as just described. E(8–11) is incremented, a word is loaded into ST, the address in D is increased by 8, and storage requests are generated when needed. When E(8–11) and E(12–15) are equal, the last word is loaded into main storage and an end-op cycle is taken.

If D(21) = 0, and if E(8–11) and E(12–15) are not equal, the first word is to be placed into the left word of the doubleword location in main storage and the second word into the right word.

SHIFT

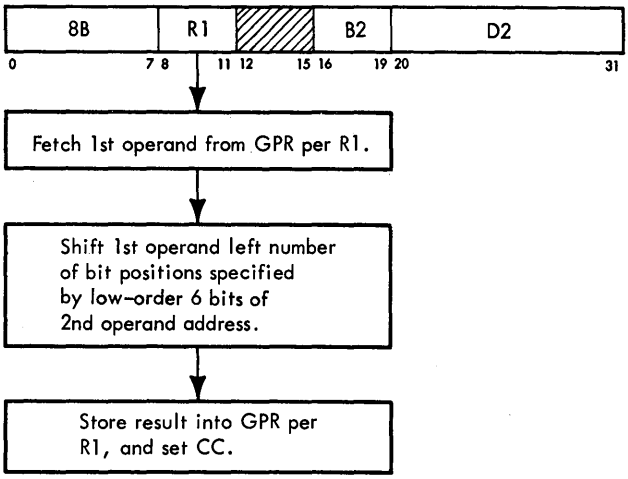
There are four fixed-point shift instructions: Shift Left Single, SLA; Shift Left Double, SLDA; Shift Right Single, SRA; and Shift Right Double, SRDA. Their function is to shift the first operand (right or left) and to store the result into the first operand address. The first operand may be a word or a doubleword in length and is shifted the amount specified by the low-order six bits of the second operand address. The specified amount of shift is accomplished in increments of left 1, left 2, left 4, and right 4 shifts. To expedite the operation, combinations of these increments are used whenever possible, and a maximum number of left 4 and right 4 shifts are used.

The CC is set to indicate the status of the result. A left-shifted result is tested for an overflow condition to determine if significant high-order digits were lost. During a right-shift, significant low-order digits may be lost with no indication.

Shift Left Single, SLA (8B)

- Shift 1st operand (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd operand address and place result into 1st operand location.

- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- Methods of shifting:
  Left 1 from T to PAA.
  Left 2 from AB to PAB.
  Left 4 from PAA or PAB to PAL.
- E(12–15) indicates number of left 4 shifts to be performed.
- E(12–15) is reduced by 1 after each multiple of four shifts occurs.
- 0's are supplied to vacated bit positions.
- Overflow occurs if data is shifted out of bit position 1.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Shift Left Single, SLA, instruction shifts the first operand left the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the second operand address is ignored.) The second operand request per D, normally initiated during RS I-Fetch, is blocked during I-Fetch of a shift instruction because D does not contain a main storage address. Refer to "Basic RS and SI I-Fetch" in Section 1 of this chapter.

The sign of the first operand remains unchanged. All 31 bits of the operand participate in the left shift. Zeros are transferred into the vacated low-order register positions. If a bit unlike the sign bit is shifted out of position 1, an overflow occurs, causing a program fixed-point overflow interruption during end op if the fixed-point overflow mask bit is a 1.

Left-shifting can be accomplished by three methods: (1) shifting left 1 bit position from T to the parallel adder, (2) shifting left 2 bit positions from AB to the parallel adder, and (3) shifting left 4 bit positions from the parallel adder to PAL. In the interest of speed, the desired amount of left shift is accomplished using a maximum number of left 4 shifts and a minimum number of left 1 and left 2 shifts (not more than one of each). Also, whenever possible, a left 1 or a left 2 shift is combined with a left 4 shift because these combinations can be accomplished in one pass through the parallel adder. Table 3-7 shows how left-shifting is accomplished for any amount of shift desired.

See Diagram 5-116, FEMDM, a flowchart of the SLA instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the SLA instruction, PAL(58–63) is first tested to determine the amount of shift. (PAL contains the same value as D.) If PAL(58–61) = 0, a shift of less than 4 is needed; PAL(62) is then tested. If PAL(62) = 1, a shift of either 3 or 2 is to be performed; if PAL(62) = 0, either a shift of 1 or no shift is to be performed.

As an example, assume that PAL(58–61) does not equal 0 and that PAL(62,63) = 11. A shift of left 7, left 11, or more is indicated. STAT C is set to S(0), the sign of the first operand that will be transferred to LS at the end of the operation. D(18–21), which is equal to PAL(58–61), is transferred to E(12–15). This value is reduced sequentially by 1 after every left 4 shift until it equals 0001, at which time no more shifting of data is necessary. The contents of T are transferred to PAA(31–62), thus achieving a left 1 shift. T(32) is propagated into PAL(26–31). Because a left 4 shift takes place only when transferring data from PAA or PAB into PAL, PAA(8–67) is shifted left 4 positions into PAL(4–63). PAL(26–32) is now tested to see that no important data has been lost during the shifting. If these bits are not all 0's or all 1's, STAT B is set, indicating overflow occured; if found set during end op, a program fixed-point overflow interruption occurs if the associated overflow mask bit in the PSW is a 1. PAL(24–67) is transferred to AB(24–67). E(12–15) is tested to see whether it contains a value of 0001. If it does, only a left 7 shift is to be performed; otherwise, a shift of left 11 or more is to be made.

Assume E(12–15) = 0001. A 1 is subtracted from E(12–15). The contents of AB(6–67) are transferred to PAB(4–65). This transfer accomplishes a left 2 shift, making a total left shift of 7 positions. PAL(26–32) is tested for overflow, and PAL(32–63) is tested for all 0's. The data is transferred to LS, and an end-op cycle is taken to complete the operation.

Now assume E(12–15) does not equal 0001, indicating a total left shift of at least 11 positions or more is called for by the instruction. Because at this point the data has

Table 3-7. Left Shift Combinations

PAL(58–61)	PAL(62,63)	Total Shift Desired	Incremental Shifting Sequence		
0000	00	None	None		
	01	Left 1	Left 1		
	10	Left 2	Left 2		
	11	Left 3	Left 1	Left 2	
Not 0000	00	Left 4, 8, 12 and so on	Left 4	Left 4 until E(12–15) = 1	
	01	Left 5, 9, 13 and so on	Left 1 and left 4	Left 4 until E(12–15) = 1	
	10	Left 6, 10, 14 and so on	Left 2 and left 4	Left 4 until E(12–15) = 1	
	11	Left 7, 11, 15 and so on	Left 1 and left 4	Left 2 if E(12–15) = 1	
				Left 2 and left 4 if E(12–15) ≠ 1	Left 4 until E(12–15) = 1

already been shifted left 5 positions, a left 6 shift is necessary to achieve the minimum left 11 shift. A 1 is subtracted from E(12–15). AB(6–67) is transferred to PAB(4–65), yielding a left 2 shift. A left 4 shift is achieved by shifting the contents of PAB(4–65) to PAL(0–61). PAL(26–32) is tested for overflow, and PAL(32–63) is tested for all 0's. PAL(32–63) is transferred to T(32–63), and T(32–63) is transferred to the GPR per E(8–11). The value of STAT C is placed into the sign position of the GPR. E(12–15) is again checked for a value of 0001. If the bits now contain this value, the data has been shifted the correct number of times. A 1 is subtracted from E(12–15), and an end-op cycle is taken, completing the operation. If E(12–15) does not equal 0001, a left shift of more than 11 is required for this instruction.

If E(12–15) still does not equal 0001, 1 is again subtracted from E(12–15). T(32–63) is transferred to PAA(32–63), and the data is shifted left 4 positions into PAL(28–63). After testing PAL(26–32) and PAL(32–63), the contents of PAL(32–63) are transferred to T, and the data in T(33–63) is transferred into the GPR per E(8–11). The value of STAT C is placed into the GPR sign position per E(8–11).

Note: The result of the first 11 shifts is transferred to LS before E(12–15) is tested to see whether a shift of more than 11 places is required. If a total shift greater than 11 is specified, the data is shifted left an additional four places and transferred to LS, where it destroys the 11-place shifted operand stored earlier. E(12–15) is tested; if a shift greater than 15 in specified, the data is again shifted left 4 and transferred to LS. This procedure of testing E(12–15), shifting left 4, and transferring the result to LS continues until E(12–15) equals 0001.

E(12–15) is again checked for 0001. If E(12–15) = 0001 at this time, 1 is subtracted from it and an end-op cycle is taken, completing the operation. If E(12–15) does not equal 0001, the operand continues to be shifted in multiples of 4 until E(12–15) equals 0001. The CC is then set per STAT A, STAT B, and the result sign [T(32)], and an end-op cycle is taken.

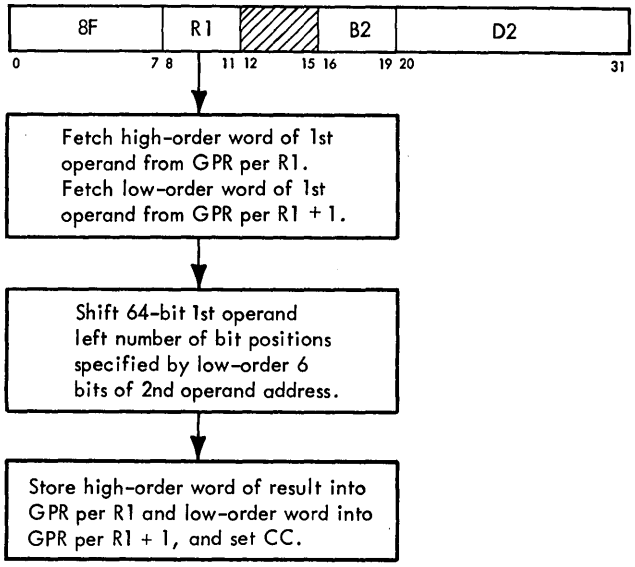
Left shifts of other amounts are performed in a similar manner. (Refer to Table 3-7 and Diagram 5-116, FEMDM.)

Shift Left Double, SLDA (8F)

- Shift 1st operand (in GPR per R1 and R1 + 1) left number of bit positions specified by low-order six bits of

2nd operand address and place result into 1st operand location.

- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- E(12–15) indicates number of left 4 shifts to be performed.
- E(12–15) is reduced by 1 after each multiple of four shifts occurs.
- High-order bits of low-order word of doubleword operand are saved and placed into high-order word operand.
- 0's are supplied to vacated bit positions.
- Overflow occurs if data is shifted out of bit position 1 of high-order word.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Shift Left Double, SLDA, instruction shifts the doubleword first operand left the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction is the address of the GPR containing the high-order 32 bits of the doubleword operand. The low-order word of the doubleword operand is in the GPR per R1 + 1. The R1 field of the instruction must specify an even register. When R1 is odd, a program specification interruption occurs.

The sign of the doubleword operand is the sign of the even GPR. The high-order bit (sign) of the odd GPR is treated as an integer bit. As the information is shifted, 0's are supplied to the vacated low-order positions. If a bit unlike the sign bit is shifted out of bit position 1 of the high-order word of the doubleword operand, an overflow occurs. The overflow causes a program fixed-point overflow interruption if the associated mask bit in the PSW is a 1.

The SLDA instruction is similar to the SLA instruction in that a left 1 shift occurs when transferring data from T to PAA, a left 2 shift occurs when transferring data from AB to PAB, and a left 4 shift occurs when data is transferred from the inputs of the parallel adder to PAL. Also, the total shift specified is accomplished using a maximum number of left 4 shifts and a minimum number of left 1 and left 2 shifts. The differences are a result of handling a doubleword, one word at a time, in the SLDA instruction.

B(64–67), which contains the overflow bits from the low-order word, may be thought of as a four-bit register inserted between the high-order word and the low-order word. It is therefore necessary to shift these bits four bit positions to the left to position them correctly into the high-order word. This positioning is accomplished in different ways according to the shift being performed; the result, however, is always that of shifting the overflow bits left four bit positions.

Refer to Diagram 5-117, FEMDM, a flowchart of the SLDA instruction. At the start of execution, the first 16 bits of the instruction are in E, the high-order word of the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the beginning of the operation, the sign of the operand (which will be transferred to LS at the end of the operation) is stored into STAT C. Because the low-order word of the doubleword operand is treated first, that word is now transferred from LS. (Recall that S and T presently contain the high-order word of the doubleword.) Once the low-order word of the doubleword is obtained, the data is shifted as for the SLA instruction. The high-order bits of the low-order word of the doubleword are not lost when shifted out but are transferred to B(64–67). When the high-order word of the doubleword is obtained, these bits are shifted into the high-order word of the operand.

The low-order word is shifted the amount specified by PAL(62,63), and the low-order 32 bits of the result are stored in the GPR specified by R1 + 1 (odd register). A zero test is performed, and STAT A is set if the low-order word is all zeros. The high-order bits that were shifted out of the low-order word are gated from PAL(28–31) to B(64–67) where they are stored, shifted left four positions and appended to the high-order word. The high-order word is transferred from the even GPR specified by R1, is shifted the same amount as the low-order word per PAL(62,63), and is added to the overflow bits from the low-order word.

An overflow test is performed to determine if significant bits were lost from the high-order word, and STAT B is set if an overflow occurred. A zero test is performed, and STAT A is reset if the high-order result is not all zeros.

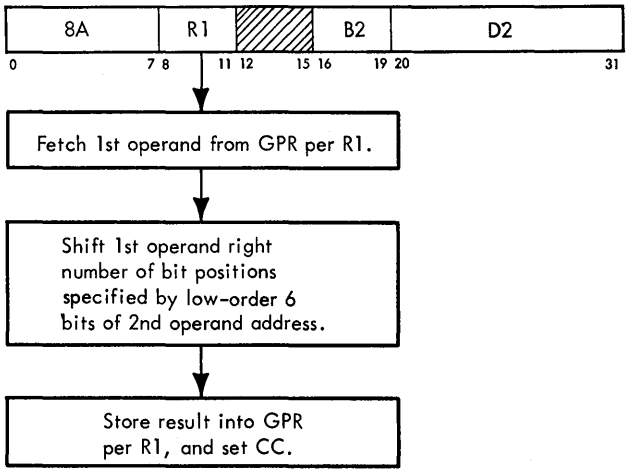
If PAL(58–61) was zero at the end of I-Fetch, the last shift has been performed and the low-order 31 bits of the high-order word, plus the inserted sign bit (per STAT C), are stored into the even GPR specified by R1. The CC is set per hardware conditions, and an end-op cycle is taken.

If PAL(58–61) was not zero, the low-order word has been transferred from the odd GPR to S, and one or more left 4 shifts remain to be performed. The low-order word in S is shifted left 4 and stored into S. The four overflow bits are retained in B(64–67), the high-order word in T is gated to PAA, B(64–67) is gated to PAB, and the result is shifted left 4 to PAL. This sequence shifts the high-order word, plus the overflow bits from the low-order word, left 4. The low-order 31 bits of the high-order word, plus the inserted sign bit (per STAT C), are gated to T and stored into the even GPR per R1. The number of left 4 shifts to be performed is determined by E(12–15), which was set per D(18–21) at the start of execution. E(12–15) is decremented after each left 4 shift is performed. The microprogram remains in the shift left 4 loop until E(12–15) is reduced to 0001, at which time the low-order word is stored into the odd GPR specified by R1 + 1. The CC is set per STAT A, STAT B, and the result sign [A(0)], and an end-op cycle is taken.

When the low-order word is shifted, a zero test is performed and STAT A is set if the low-order result is all zeros. When the high-order word is shifted, a zero test is performed and STAT A is reset if the high-order result is not all zeros. When the high-order word is shifted, an overflow test is performed; STAT B is set if an overflow occurred and a program fixed-point overflow interruption is taken if the associated mask bit in the PSW is a 1.

Shift Right Single, SRA (8A)

- Shift 1st operand (in GPR per R1) right number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format: (shown in adjacent column)



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- Method of shifting: right 4 from PAA or PAB to PAL.
- Shifts of right 3 or less are obtained by combining left 1, left 2, or left 3 shifts with right 4 shift.
- E(12–15) indicates number of shifts to be performed in multiples of 4.
- E(12–15) is reduced by 1 after each shift of 4.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Shift Right Single, SRA, instruction shifts the first operand right the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the address is ignored.)

The sign of the first operand remains unchanged. All 31 bits of the operand participate in the shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted without inspection and are lost.

Right-shifting is accomplished only by shifting right 4 bit positions at a time from the parallel adder to the PAL. Right shifts of less than 4 are obtained by combining left 1, left 2, or left 3 shifts with the right 4 shift. In the interest of speed, the desired amount of right shift is accomplished using a maximum number of right 4 shifts and a minimum number of left 1 and left 2 shifts (not more than one of each). Also, wherever possible, a left 1 or left 2 shift is combined with a right 4 shift because these combinations can be accomplished in one pass through the parallel adder. Table 3-8 shows how right-shifting is accomplished for any amount of shift desired.

Diagram 5-118, FEMDM, is a flowchart of the SRA operation. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the SRA operation, PAL(58–63) initially determines the amount of shift. PAL(58–61) determines whether a shift of more than right 3 is to occur. If PAL(58–61) = 0, a shift of right 3 or less is to be performed; if PAL(58–61) does not equal 0, then shifts of right 4 or more are to occur.

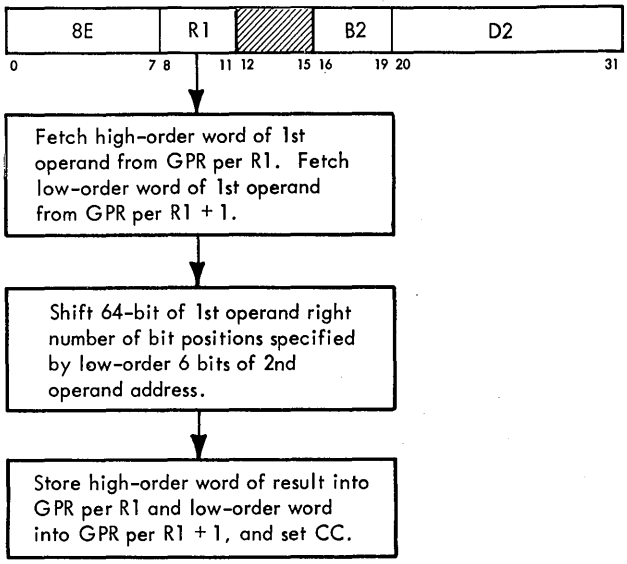
As an example, assume that PAL(58–61) does not equal 0 and that PAL(62,63) = 01. A shift of right 5, right 9, or more is indicated. D(18–21) is transferred to E(12–15) to determine the number of right 4 shifts to be used if a shift of more than right 5 is to occur. PAL(32–63) is now tested for all 0's. If this condition is present, STAT A is set. The first operand in T(32–63) is transferred to AB, with T(32) propagated into A(26–31).

The first operand in T(32–63) is transferred to PAA(31–62), causing a left 1 shift of the data, and T(32) is propagated into PAL(26–31). PAA(31–62) and PAL(26–31) are transferred right 4 positions to PAL(35–66) and PAL(30–35), respectively, giving, in effect, a shift of right 3 positions. The data in PAL is transferred to AB(24–67), from where it is transferred to PAB(4–65), causing a left 2 shift. The total effective shift at this time is a right 1 shift. PAL(32–63) is now tested for all 0's, and STAT A is set if all 0's are present. PAL(32–63) is transferred to T(32–63), from where the data is transferred into the GPR per E(8–11). E(12–15) is tested for 0001. If this value exists, an end-op cycle is taken to complete the operation. If E(12–15) does not equal 0001, then the data continues to be shifted in multiples of 4, and E(12–15) is reduced by 1 for each right 4 shift until it equals 0001. At this time the CC is set per STAT A and the result sign [T(32)], and an end-op cycle is taken.

Right shifts of other amounts are performed in a similar manner. (Refer to Table 3-8 and Diagram 5-118, FEMDM.)

Shift Right Double, SRDA (8E)

- Shift 1st operand (in GPR per R1 and R1 + 1) right number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.

Table 3-8. Right Shift Combinations

PAL(58–61)	PAL(62,63)	Total Shift Desired	Incremental Shifting Sequence		
0000	00	None	None		
	01	Right 1	Left 1 and right 4	Left 2	
	10	Right 2	Left 2 and right 4		
	11	Right 3	Left 1 and right 4		
Not 0000	00	Right 4, 8, 12 and so on	Right 4	Right 4 until E(12–15) = 1	
	01	Right 5, 9, 13 and so on	Left 1 and right 4	Left 2 and right 4	Right 4 until E(12–15) = 1
	10	Right 6, 10, 14 and so on	Left 2 and right 4	Right 4	Right 4 until E(12–15) = 1
	11	Right 7, 11, 15 and so on	Left 1 and right 4	Right 4	Right 4 until E(12–15) = 1

- D(18–23) indicates total amount of shift.
- E(12–15) indicates number of shifts to be performed in multiples of 4.
- E(12–15) is reduced by 1 after each shift of 4.
- Low-order bits of high-order word of doubleword operand are saved and placed into low-order word of operand.
- Value of operand sign is supplied to vacated bit positions.
- CC settings:
  Result in PAL equals zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Shift Right Double, SRDA, instruction right-shifts the doubleword first operand the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the address is ignored.) The R1 field of the instruction addresses the high-order 32 bits of the doubleword operand. The low-order word of the operand is in the GPR per R1 + 1. The R1 field must specify an even GPR; a program specification interruption occurs when R1 is odd.

The sign of the doubleword operand is the sign of the even GPR (R1). The sign bit (high-order bit) of the odd GPR (R1 + 1) is treated as an integer bit. Bits equal to the sign of the doubleword operand are supplied to the vacated high-order positions as the information is shifted. Bits shifted out of the low-order positions are lost.

Diagram 5-119, FEMDM, is a flowchart of the SRDA instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

The SRDA instruction execution is similar to that of the SRA instruction. The differences are a result of shifting a doubleword, one word at a time, in the SRDA instruction. Because the high-order word is shifted first, it is necessary to retain the underflow bits from the high-order word, to shift them right 4, and to append them to the low-order word.

Any specified amount of right shift involves at least one shift of right 4 for both the high-order word and the low-order word. Therefore, the underflow bits from the high-order word are appended to the low-order word [by transferring B(64–67) to PAB(28–31)] before the right 4 shift. This action results in the underflow bits and the low-order word being shifted right 4 as a unit.

Each time the shifted high-order word is stored, a zero test is performed and STAT A is set if the result is 0. When the low-order word has been shifted the same amount, STAT A is reset if the result is not 0. STAT A and the result sign [A(0)] determine the CC.

This section discusses the 44 instructions making up the floating-point instruction set. Before analyzing the instructions, however, the following paragraphs discuss exponent overflow and underflow and zero results, and list the conditions at the start of execution. (For a discussion of data formats, normalization, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.)

EXPONENT OVERFLOW AND UNDERFLOW

- Exponent overflow occurs if two positive characteristics are added, or if positive number is added to positive characteristic, and final result is negative characteristic.
- Exponent underflow occurs if two negative characteristics are added, or if quantity is subtracted (complement added) from negative characteristic, and final result is positive characteristic.
- Exponent underflow program interruption occurs if PSW(38) = 1.

During floating-point operations, values may be chosen that cause the CE to yield invalid results. For example, the largest positive exponent that can be expressed as a floating-point characteristic is +63, and is represented in excess-64 notation as 111 1111 (7F, hex). Assume that +63 is the characteristic of a floating-point operand and that a 1 is added to it:

```
0 1 2 3 4 5 6 7
S 1 1 1 1 1 1 1     +63 (Excess-64 Notation)
S 0 0 0 0 0 0 1     +1
S 0 0 0 0 0 0 0
  ↓
Carry
```

Note that the sum in bits 1–7, instead of indicating an exponent of +64, indicates an exponent of -64, 128 less than the true exponent. Thus, exponent overflow occurred. The rule for exponent overflow is: if two positive characteristics are added, or if a positive number is added to a positive characteristic, and the final result is a negative characteristic, exponent overflow occurred. This rule does not hold for intermediate result characteristics which may exceed the highest expressible exponent.

If exponent overflow occurs, an interruption is forced and cannot be masked off (refer to Chapter 1). The resulting invalid characteristic is not altered and remains in the result register for examination by the interruption-handling microprogram.

The largest negative exponent that can be expressed as a floating-point characteristic is -64, and is represented in excess-64 notation as a characteristic of all zeros. Assume that -64 is the characteristic of a floating-point operand and that a 1 is subtracted from it:

```
0 1 2 3 4 5 6 7
S 0 0 0 0 0 0 0     –64 (Excess-64 Notation)
S 1 1 1 1 1 1 1     2's Complement of 1
S 1 1 1 1 1 1 1
```

The difference in bits 1–7, instead of indicating an exponent of -65, indicates an exponent of +63, 128 more than the true exponent. This is known as exponent underflow. The rule for exponent underflow is: if two negative characteristics are added, or if a quantity is subtracted (complement-added) from a negative characteristic, and the final result is a positive characteristic, exponent underflow occurred. This rule does not hold for intermediate characteristics which may exceed the most negative expressible exponent.

If exponent underflow occurs, an interruption takes place only if the exponent-underflow mask bit [PSW(38)] is a 1 (refer to Chapter 1). If the interruption is taken, the resulting invalid characteristic is not altered and remains in the result register for examination by the interruption-handling microprogram. However, if the underflow interruption is masked off, the entire result (sign, characteristic, and fraction) is converted to a true zero (see "Zero Results"). This value can be a valid result for some calculations because exponent underflow indicates that the result was very small (less than 16^{-64}) and therefore close to zero.

Referring to the two examples given above, the CE determines if exponent overflow or underflow occurred by examining bit 1 of the final characteristic and testing for a carry out of bit 1. If a carry occurred during a characteristic addition and bit 1 is a 0, exponent overflow occurred; if a carry did not occur during a characteristic subtraction and bit 1 is a 1, exponent underflow occurred.

ZERO RESULTS

A zero result is normally stored into the result register as a true zero; that is, a zero characteristic, a zero fraction, and a plus sign. A true zero may occur because of the magnitude of the operands or it may be forced. A true zero

is forced if exponent underflow occurs during add, subtract, multiply, or divide instructions and the exponent-underflow mask is off [PSW(38) = 0]. A true zero is also forced when a result fraction is zero and the program interruption for significance is masked off during add or subtract instructions. "Significance" means that an add or subtract instruction resulted in a zero fraction. This condition causes a significance program interruption if the significance mask is on [PSW(39) = 1]. When a significance condition occurs with the mask on, the result characteristic and sign remain unchanged and are stored with the zero fraction. True zero is never forced when a zero fraction occurs during a load, store, or halve instruction. Whenever a result has a zero fraction, an exponent overflow or exponent underflow condition is ignored.

CONDITIONS AT START OF EXECUTION

The conditions at the start of execution for the RR and RX instructions, short and long operands, are:

1. RR, Short Operands:
   a. First operand is in A, B, and D (24-bit fraction only).
   b. Second operand is in S and T.
   c. Instruction is in E.
2. RR, Long Operands:
   a. 32 bits of first operand are in A, B, and D (24-bit fraction only).
   b. 32 bits of second operand are in S and T.
   c. Low-order fractions of first and second operands are in LS.
   d. Instruction is in E.
3. RX, Short Operands:
   a. First operand is in S and T.
   b. Main storage request for second operand has been issued per D.
   c. First 16 bits of instruction are in E.
4. RX, Long Operands:
   a. 32 bits of first operand are in S and T.
   b. Low-order fraction of first operand is in LS.
   c. Main storage request for 2nd operand has been issued per D.
   d. First 16 bits of instruction are in E.

The subsequent paragraphs describe the execution sequences for floating-point instructions. All figures and diagrams supporting the text abbreviate the word "characteristic" to "charistic."

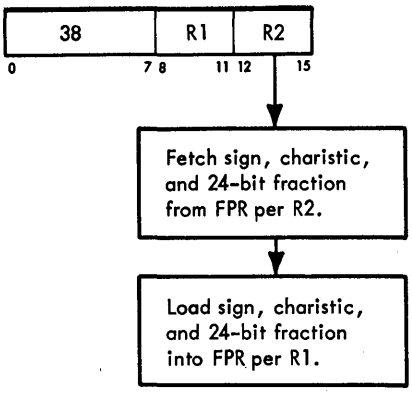
LOAD

The floating-point load instructions provide a means of loading operands into the LS FPR's. The load operation may be register-to-register (RR format) or storage-to-register (RX format) and may use short or long operands. In any case, the instruction loads the second operand into the first operand location, and the second operand location remains unchanged.

In addition, certain floating-point load instructions can test or modify the sign of the second operand before loading it into LS. The second operand may be also complemented before loading.

Load, LER (38) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1).
- RR format:

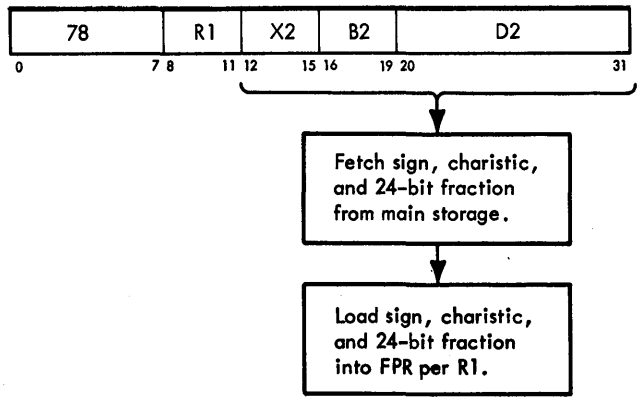


- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Load, LER, instruction (Diagram 5-202, FEMDM) loads the 32-bit second operand from the LS FPR specified by the R2 field into the first operand location specified by the R1 field. During the RR I-Fetch, the second operand is gated to S and T. At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the contents of T (second operand) are gated to the LS FPR specified by R1. If a specification check did occur, the operation is suppressed and a specification program interruption occurs.

Load, LE (78) – RX Short Operands

- Load 2nd operand (in storage) into 1st operand location (in FPR, per R1).
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

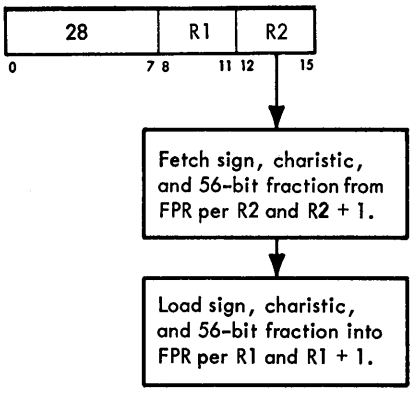
The Load, LE, instruction (Diagram 5-203, FEMDM) loads the 32-bit second operand from the main storage location specified by the effective address into the first operand location specified by the R1 field. The effective address of the second operand must be on a word boundary or a specification program interruption occurs. During the RX I-Fetch, the effective address is computed and placed into D. A main storage request for the second operand is then initiated per D.

At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, either SDBO(0–31) or SDBO(32–63) is gated to T. Because a main storage request is always for a doubleword and only a word operand is desired, D(21) determines which word of the main storage doubleword is used; if D(21) = 1, the right word is gated to T; if D(21) = 0, the left word. From T, the second operand is gated to the LS FPR specified by R1.

If a specification check occured at the start of the execution phase, the operation is suppressed and a specification program interruption occurs.

Load, LDR (28) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:



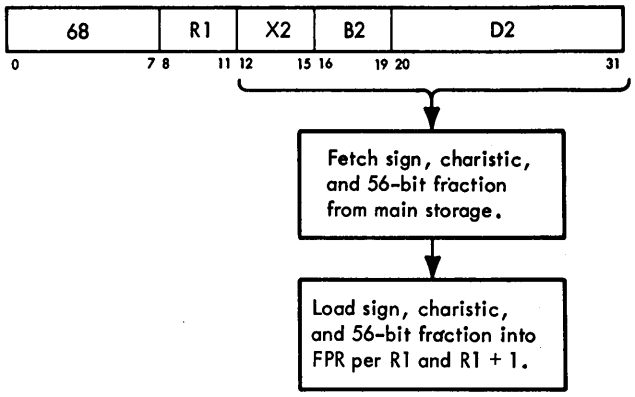
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The load, LDR, instruction (Diagram 5-202, FEMDM) loads the doubleword second operand from the LS FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the RR I-Fetch, the high-order 32 bits of the second operand are placed into S and T. At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the low-order 32 bits of the second operand are fetched from the odd register of the even/odd pair of FPR's specified by the R2 field. From T, the low-order 32 bits of the second operand are loaded into the odd register of the even/odd pair of FPR's specified by the R1 field. The high-order 32 bits of the second operand are then gated from S to T. From T, they are loaded into the even register of the even/odd pair of FPR's specified by the R1 field.

If a specification check occurred at the start of the execution phase, the operation is suppressed and a specification program interruption occurs.

Load, LD (68) – RX Long Operands

- Load 2nd operand (in storage) into 1st operand location (in FPR, per R1 and R1 + 1).
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Load, LD, instruction (Diagram 5-203, FEMDM) loads the doubleword second operand from the main storage location specified by the effective address into the first operand location specified by R1 and R1 + 1. The effective address of the second operand must be on a doubleword boundary or a specification program interruption occurs. During the RX I-Fetch, the effective address is computed and placed into D. A main storage request for the second operand is then initiated per D.

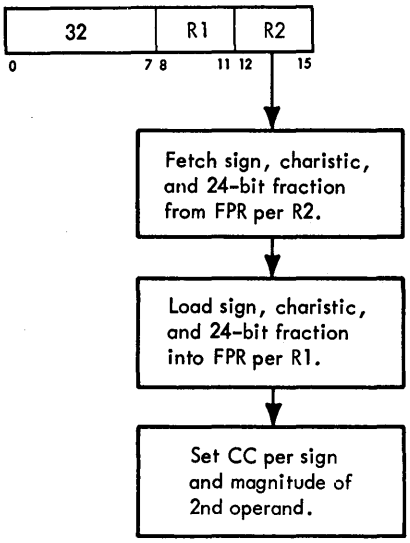
At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the doubleword second operand is gated from SDBO to ST.

The low-order 32 bits are then loaded into the odd register of an even/odd pair of FPR's specified by the R1 field. The high-order 32-bits of the second operand are then gated from S to T. From T, they are loaded into the even register of the even/odd pair of FPR's specified by the R1 field.

If a specification check occurred at the start of the execution phase, the operation is suppressed, and a specification program interruption occurs.

Load and Test, LTER (32) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1).
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.

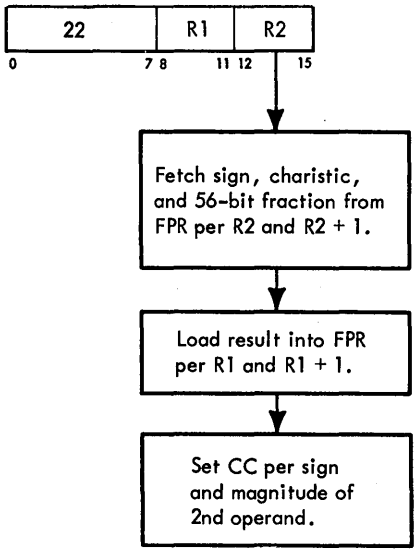
The Load and Test, LTER, instruction (Diagram 5-204, FEMDM) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. The sign and magnitude of the second operand determine the CC, as follows:

1. If the fraction of the second operand equals zero, the CC is set to 0.
2. If the second operand (sign, characteristic, and fraction) is less than zero, the CC is set to 1.
3. If the second operand (sign, characteristic, and fraction) is greater than zero, the CC is set to 2.

The LTER execution is similar to LER execution. However, the sign of the second operand is saved in STAT C, and STAT A is set if the fraction equals zero. The CC is determined during the normal end-op cycle. If STAT A is set, indicating that the fraction equals zero, the CC is set to 0. The sign and characteristic are not considered when the fraction equals zero. If the second operand fraction is not equal to zero, the sign (STAT C) determines a greater-than-zero or less-than-zero condition. If the sign is minus (STAT C set), the second operand is less than zero and the CC is set to 1. If the sign is plus (STAT C reset), the second operand is greater than zero and the CC is set to 2. Setting the CC depends upon the 'Set-CR' micro-order, the instruction op-code, and the hardware conditions specified above.

Load and Test, LTDR (22) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.

The Load and Test, LTDR, instruction (Diagram 5-205, FEMDM) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. The sign and magnitude of the second operand determine the CC, as follows:

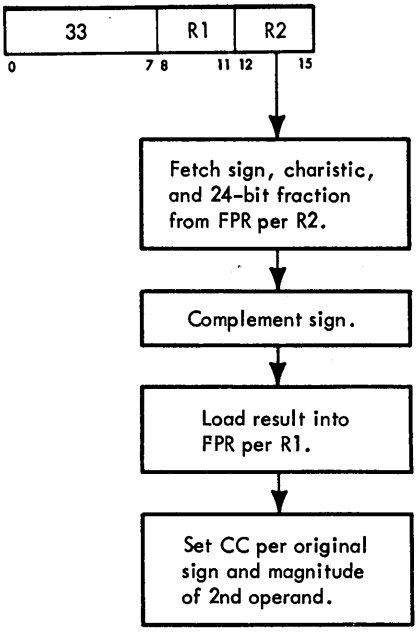
1. If the fraction of the second operand equals zero, the CC is set to 0.

2. If the second operand (sign, characteristic, and fraction) is less than zero, the CC is set to 1.
3. If the second operand (sign, characteristic, and fraction) is greater than zero, the CC is set to 2.

The LTDR execution is similar to LDR execution. However, the sign of the second operand is saved in STAT C, and STAT A is set if the fraction equals zero. The CC is determined during the normal end-op cycle in the same manner as explained for the LTER instruction.

Load Complement, LCER (33) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign complemented.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  Original sign is plus: CC = 1.
  Original sign is minus: CC = 2.

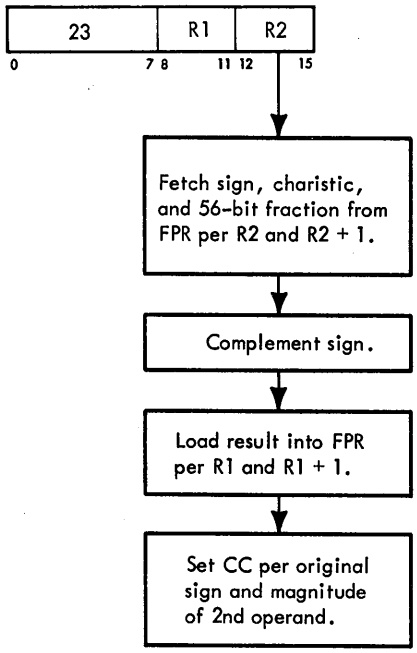
The Load Complement, LCER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is changed to the opposite value (complemented). The original sign and magnitude of the second operand determine the CC, as follows:

1. If the fraction of the second operand equals zero, the CC is set to 0.
2. If the original sign of the second operand is plus, the CC is set to 1.
3. If the original sign of the second operand is minus, the CC is set to 2.

Except for complementing the sign of the second operand, LCER execution is identical to LTER execution.

Load Complement, LCDR (23) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign complemented.
- RR format:



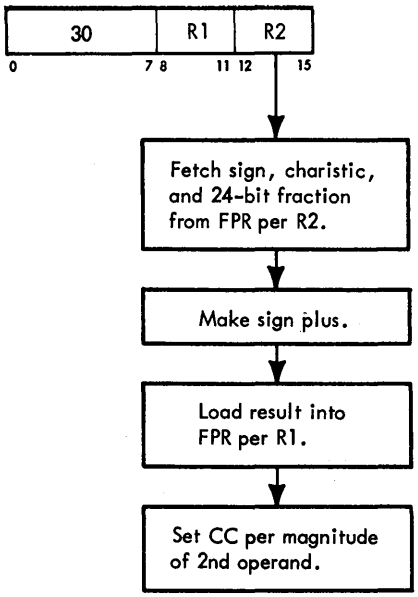
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  Original sign is plus: CC = 1.
  Original sign is minus: CC = 2.

The Load Complement, LCDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is complemented. The original sign and magnitude of the second operand determine the CC in the same manner as explained for the LCER instruction.

Except for complementing the sign of the second operand, LCDR execution is identical to LTDR execution.

Load Positive, LPER (30) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign made plus.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is greater than zero: CC = 2.

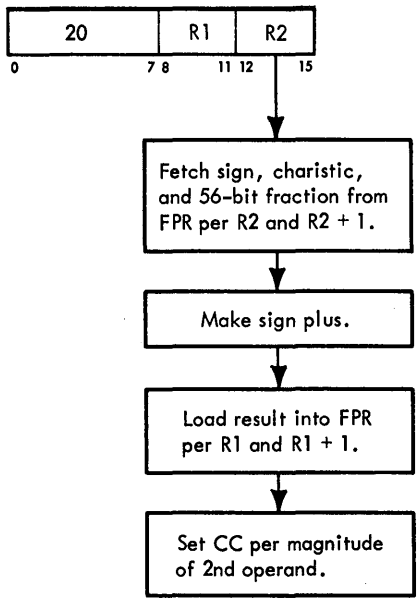
The Load Positive, LPER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is made plus. Thus, the result stored is always zero or greater. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC, as follows:

1. If the second operand fraction equals zero, the CC is set to 0.
2. If the second operand is greater than zero, the CC is set to 2.

Except for making the sign of the second operand plus, LPER execution is identical to LTER execution.

Load Positive, LPDR (20) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign made plus.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is greater than zero: CC = 2.

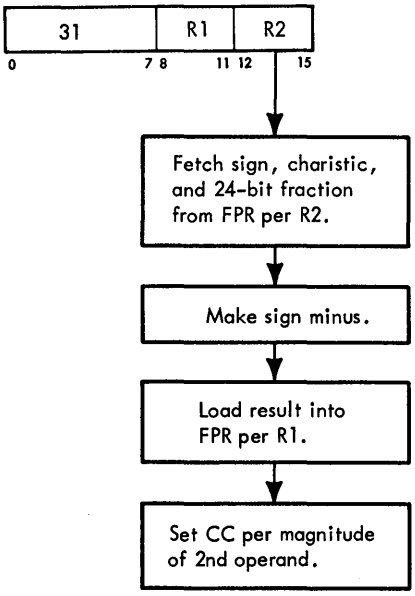
The Load Positive, LPDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is made plus. Thus, the result stored is always zero or greater. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC in the same manner as explained for the LPER instruction.

Except for making the sign of the second operand plus, LPDR execution is identical to LTDR execution.

Load Negative, LNER (31) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign made minus.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.

The Load Negative, LNER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is made minus. Thus, the result stored is always zero or less. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC, as follows:

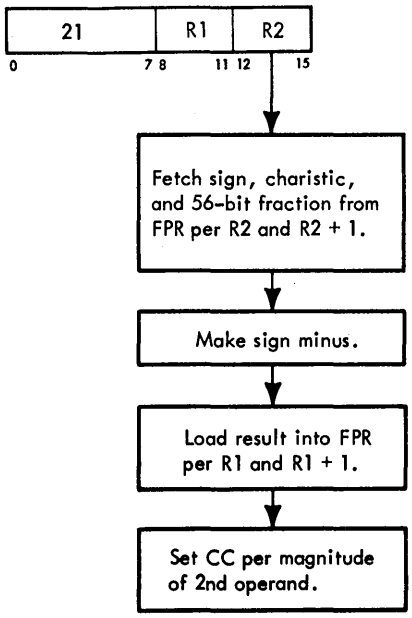
1. If the second operand fraction equals zero, the CC is set to 0.
2. If the second operand is less than zero, the CC is set to 1.

Except for making the sign of the second operand minus, LNER execution is identical to LTER execution.

Load Negative, LNDR (21) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign made minus.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.

The Load Negative, LNDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is made minus. Thus, the result stored is always zero or less. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC in the same manner as explained for the LNER instruction.

Except for making the sign of the second operand minus, LNDR execution is identical to LTDR execution.

ADD, SUBTRACT, AND COMPARE

- 20 add, subtract, and compare instructions.
- Short and long operands are available in both formats.
- Add and subtract instruction results may be normalized or unnormalized.
- Results are in true form with plus or minus values.

The 20 floating-point add-type instructions are divided into three major groups: add, subtract, and compare. RR and RX formats using short and long operands are available in each group. The results of add and subtract instruction may be normalized or unnormalized, depending upon the instruction being executed. A CC is set on all add-type instructions; the compare instructions cause a CC to be set with no result stored.

The CE computes the sum of floating-point numbers as follows (assume that $0.004_{16} \times 16^5$ is to be added to $0.502_{16} \times 16^4$.):

1. Equalizes the characteristics.
   a. If the characteristics are unequal, the operand with the smallest characteristic is shifted right the number of hex digits necessary to equalize the characteristics. In the example, $0.502_{16} \times 16^4$ has the smallest characteristic (exponent); it is therefore changed to $0.0502_{16} \times 16^5$, thus making the characteristics equal.
   b. If the number of shifts exceeds the number of hex digits available, the operand with the largest characteristic becomes the intermediate result.
2. When the characteristics are equal, algebraically adds the first and second operand fractions.
   a. If the signs are alike, adds the first operand fraction to the second operand fraction. In the example, the signs are alike; therefore, the fractions are added giving a sum of $0.0542_{16} \times 16^5$
   b. If the signs are unlike, subtracts the second operand fraction from the first operand fraction (adds the 2's complement of the second operand fraction to the first operand fraction).
3. If the intermediate result fraction is in complement form, recomplements it (takes 2's complement) to obtain the true fraction value.
4. Normalizes the intermediate result, if normalization is called for. Assume a 3-digit machine. If normalization is specified, the final result of the example (2a above) becomes $0.542_{16} \times 16^4$. If normalization is not specified, the low-order digit (guard digit) is truncated, and the final result is $0.054_{16} \times 16^5$.
5. Determines the sign and characteristic value.
6. Stores the sign, characteristic, and fraction into LS as specified by the R1 field.
7. Sets the CC per hardware conditions.

For subtraction of floating-point numbers, the algebraic rule applies: to subtract two numbers, change the sign of the subtrahend and proceed as in addition. When subtracting floating-point numbers, the sign of the second operand is complemented. The rules of addition apply as outlined in the previous paragraph. To illustrate, assume the same numbers as used above and that the signs are unlike. Thus, a 2's complement add (step 2b above) is performed:

$$\begin{array}{ll} .0502_{16} & \\ .FFC0_{16} & \text{(2's complement of } .0040_{16}\text{)} \\ \hline .04C2_{16} & \end{array}$$

The difference, then, is $0.04C_{16} \times 16^5$ if unnormalized; or $0.4C2_{16} \times 16^4$, if normalized.

The compare instructions are similar to the subtract instructions; the results, however, are not stored. The compare instructions algebraically compare the first operand with the second operand and set the CC accordingly. These objectives are accomplished by complementing the sign, algebraically adding the fractions, determining a high, low, or equal condition, and setting the CC.

The basic objectives of the add-type instructions are shown in Sheet 1 of Diagrams 5-206 and 5-207, FEMDM (short and long operands, respectively). After the RR or RX I-Fetch and the specification test, the remaining operand and/or low-order fraction(s) must be fetched or the low-order fractions reset to zeros. The signs are saved in STAT's. For short operand instructions (Diagram 5-206), zeros are gated to the low-order fractions of the 64-bit operands.

The characteristics are then compared. Preshifting occurs, if necessary, followed by the addition or subtraction of the fractions. Because the characteristics must be equal before algebraically adding the operands, the characteristics are subtracted to determine whether they are equal and whether preshifting is meaningful. For short operands, the characteristic difference must be 7 or less; for long operands, 15 or less.

If the characteristic difference is greater than 7 (short operands) or 15 (long operands) the fraction resulting after right-shifting equals zero. Therefore, preshifting is not performed, and the operand with the largest characteristic becomes the result. If preshifting is meaningless, the value in AB or ST is the result. If the characteristics are within range, the smallest fraction is right-shifted until the characteristics are equal; the fractions are then added algebraically to form an intermediate result. If a high-order carry occurs as a result of the addition (overflow), the intermediate result is right-shifted one hex digit and the characteristic is increased by 1. If this increase causes a characteristic overflow, an exponent-overflow program interruption occurs.

The intermediate result consists of 7 or 15 hex digits and a possible carry. The low-order digit is a guard digit retained from the fraction which is shifted right. Only one guard digit participates in the fraction addition. The guard digit is zero if no shift occurs.

After the addition or subtraction, a test is made for compare instructions, normalized instructions, or unnormalized instructions. Postnormalization, recomplementation, and/or fraction overflow correction is accomplished during this phase. The final result is stored (except on compare instructions), and the CC is set according to the computed results. An end op completes instruction execution.

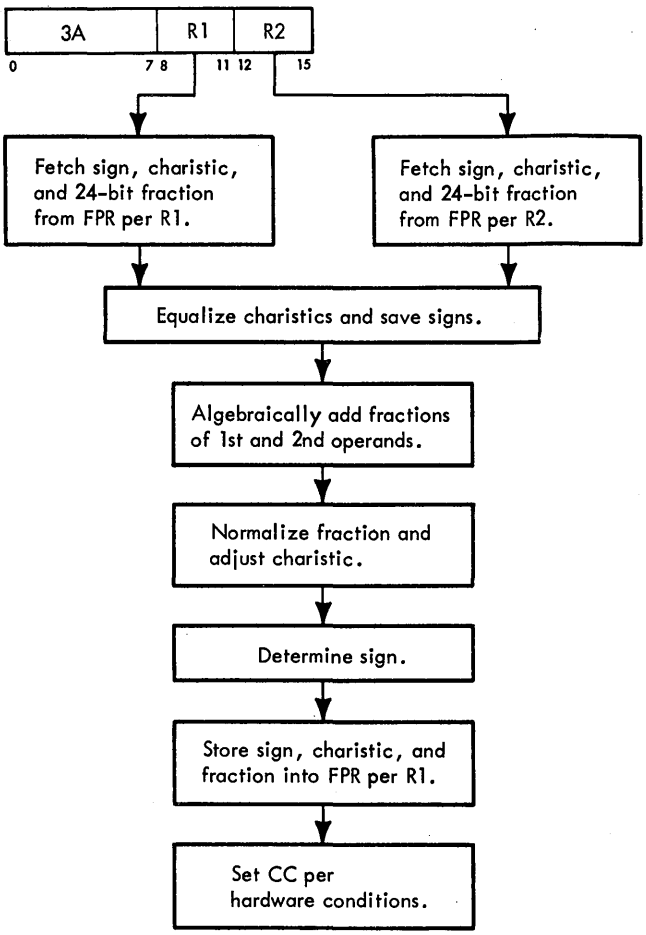
For normalized instructions the intermediate result fraction is left-shifted as necessary to form a normalized fraction. Vacated low-order digit positions are filled with zeros, and the characteristic is reduced by the amount of the shift. If normalization causes the characteristic to underflow, an exponent-underflow program interruption condition exists; the sign, characteristic, and fraction are made zero if the underflow mask bit [PSW(38)] is a 0. If PSW(38) is a 1, a program interruption occurs, and the characteristic is made 128 larger than the true result; the sign and fraction remain unchanged. If no left shift takes place, the guard digit is removed to obtain the proper fraction length.

When the intermediate result fraction is zero and the significance mask bit is a 1, a significance program interruption takes place. No normalization occurs, and the intermediate result characteristic remains unchanged. When the intermediate result is zero and the significance mask bit is a 0, a significance program interruption does not occur; rather, the characteristic and the sign are made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the result is derived algebraically. However, the sign of a zero result fraction is always positive.

Add Normalized, AER (3A) – RR Short Operands

- Algebraically add 2nd operand (in FPR, per R2) to 1st operand (in FPR, per R1) and place normalized sum into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.



The Add Normalized, AER, instruction (Diagram 5-206) algebraically adds the second operand (specified by R2) to the first operand (specified by R1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AER instruction uses 32-bit operands; therefore, the contents of the low-order halves of the FPR's in LS remain unchanged.

At the beginning of the execution phase:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC was set to 4 during I-Fetch.
4. The AER instruction is in E.

A specification test is made at the beginning of the execution phase. If a specification check exists, instruction execution is suppressed, and a specification program interruption occurs. Assume that no specification check exists.

Because the AER instruction uses short operands (32 bits) in the RR format, no operand fetch during instruction execution is necessary. The sign of the first operand is saved

in STAT F and the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic to determine the characteristic difference. Next, the second operand is gated to D and S. Because the AER instruction operates on short operands, B and T are reset, and the operand is treated as a 56-bit fraction with the low-order bits set to zero. The characteristic difference and the signs determine the next operation to be performed by means of a 10-way 'FLR' micro-order branch.

The 10-way 'FLR' branch on characteristic difference and signs occurs for all add-type instructions. When this branch is encountered, the conditions are as follows:

1. The first operand is in AB (B equals zero for short operands).
2. The second operand is in DT (T equals zero for short operands) and S.
3. The sign of the first operand is in STAT F.
4. The sign of the second operand is in STAT C.
5. The characteristic difference is in SAL and F.

To subtract the first operand characteristic from the second operand characteristic, the 2's complement of the first operand characteristic is added to the second operand characteristic. Because the serial adder consists of 8 binary bit positions, a 1 is forced into bit position 0 on the A-bus (first operand side) of the serial adder, and a 0 is forced into bit position 0 on the B-bus (second operand side) of the serial adder. The characteristic difference is then routed to SAL and F.

The 10-way 'FLR' branch is determined by the result of the characteristic subtraction and by the signs in STAT F and STAT C. The 10-way 'FLR' branch affects bits 8–11 of ROSAR as defined in Sheet 2 of Diagram 5-206. ROSAR(8–11) is set as follows:

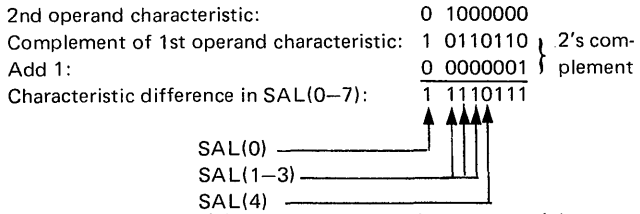
1. ROSAR(8) = 1 when a serial adder carry indicates that the second operand is greater than or equal to the first operand.
2. ROSAR(9) = 1 when the signs are alike. If the signs are alike, the fractions are added; if unlike, the fractions are subtracted.
3. ROSAR(10) = 1 when the characteristic difference is within range. ROSAR(10) equaling 1 implies that the characteristic difference is small enough so that equalizing the characteristics is meaningful. (A zero fraction may occur as a result of characteristic equalization.)
4. ROSAR(11) = 1 when the result in SAL is zero. This condition indicates that the characteristics are equal. A serial adder carry also occurs; thus ROSAR(8) will also equal 1.

Assume that the following two characteristics are to be compared to determine the ROSAR value:

2nd operand characteristic: $64_{10} = 1000000_2$

1st operand characteristic: $74_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic shown below:



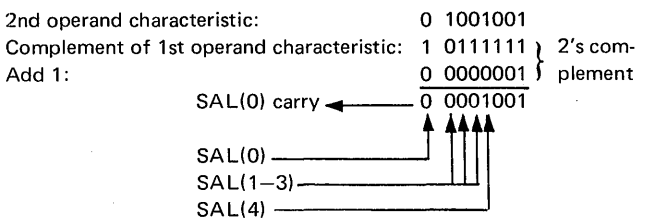
Because no SAL(0) carry occurred, ROSAR(8) = 0. ROSAR(9) depends upon the signs assigned to the fractions of the two operands. The table in Sheet 2 of Diagram 5-206 shows the bit positions tested in SAL to determine the value of ROSAR(10) and whether preshifting is meaningful. In this example, SAL(0–3) = 1111 (binary). Because a short operand instruction is being executed, ROSAR(10) = 0; therefore, the value is in AB because no carry from SAL(0) indicated that the first operand is greater than the second operand. ROSAR(11) = 0 because SAL(0–7) is not all 0's.

If all positions of SAL are 0, ROSAR(11) = 1, indicating that the characteristics of the two operands are equal. ROSAR(9) then determines the next operation (signs alike, add; signs unlike, subtract). In this case [SAL(0–7) = 0], ROSAR(8) also equals a 1 because a SAL(0) carry occurred.

For the next example, assume that the two characteristics in the previous example are interchanged. The characteristics are compared as follows:

2nd operand characteristic: 1001001
1st operand characteristic: 1000000

The first operand characteristic is subtracted from the second operand characteristic shown below:



In this example, ROSAR(8) = 1 because a SAL(0) carry occurred. ROSAR(10) = 0. Because preshifting in this example is meaningless, the operand in ST is the result fraction because the SAL(0) carry indicates that the second operand is the largest.

The two examples just discussed illustrate the determination of the ROSAR(8–11) values. Additional examples are shown in Table 3-9. ROSAR(8–11) determines the ROS branch that performs the next steps in executing the AER instruction.

Table 3-9. Examples of Branching on Characteristic Difference

Description	Example No. 1	Example No. 2	Example No. 3	Example No. 4	Example No. 5
2nd operand characteristic 1st operand characteristic	1000000 1001001	1000000 1001000	1001000 1000000	1000111 1000000	1000000 1000000
2nd operand characteristic: Complement of 1st operand characteristic Add 1 Difference in SAL(0–7)	0 1000000 1 0110110 0 0000001 1 1110111	0 1000000 1 0110111 0 0000001 1 1111000	0 1001000 1 0111111 0 0000001 ←0 0001000 SAL(0) carry	0 1000111 1 0111111 0 0000001 ←0 0000111 SAL(0) carry	0 1000000 1 0111111 0 0000001 ←0 0000000 SAL(0) carry
Within Range?* Short	No	Yes	No	Yes	Yes
Within Range?* Long	Yes	Yes	Yes	Yes	Yes
ROSAR(8–11) value	Sub 0000 Add 0100	Sub 0010 Add 0110	Sub 1000 Add 1100	Sub 1010 Add 1110	Sub 1011 Add 1111
Comments	Result in AB.	Equalize fraction in ST.	Result in ST.	Equalize fraction in AB.	Add or sub. No shift necessary.

Notes:

1. ROSAR(8) = 1 when there is a serial adder carry. A carry indicates that R2 R1.
2. ROSAR(9) = 1 when the signs are alike.
3. ROSAR(10) = 1 when the characteristics are within range. ROSAR(10) = 1 on SAL results as follows:
   a. SAL carry and SAL(0–3) = 0 and long operands.
   b. SAL carry and SAL(0–4) = 0.
   c. No SAL carry and SAL(0–3) = 1's and long operands.
   d. No SAL carry and SAL(0–4) = 1's.
4. ROSAR(11) = 1 when the SAL outputs equal 0.

The examples of determining the ROSAR(8–11) values as shown in Table 3-9 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is seven or less.

Four possible ROSAR(8–11) values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition or subtraction of fractions. For example, assume that the AER instruction requires characteristic equalization, fraction subtraction, recomplementation (second operand fraction is greater than first operand fraction), and normalization. Further, assume a ROSAR(8–11) value of 0010. The 0010 branch causes one right shift of the second operand to occur, and a 1 is added to F. Note that one guard digit is retained. SAL(4–7) is checked for 1111 (binary). When SAL(4–7) = 1111, the characteristics are equal. Because the test for a branch is made one machine cycle before the ROS branch occurs, the SAL value is one machine cycle behind the actual shift count. For this reason, a test is made for 1111 in SAL(4–7) instead of for 0000.

Once the characteristics are equal, the second operand is subtracted from the first operand (signs unlike). To subtract fractions (signs unlike), the 2's complement of the second operand fraction in DT is added to the first operand fraction in AB with the intermediate fraction result placed into AB and DT. The intermediate fraction result may be in true form or in complement form, or it may be equal to zero. If a zero fraction results, STAT A is set.

Because the larger characteristic is used as the characteristic of the result, it is gated to F(1–7); F(0) is reset.

If the second operand fraction is greater than the first operand fraction, the result is in complement form. Conversely, the result is in true form if the first operand fraction is greater than the second operand fraction. If A(7) = 1, the intermediate result is in complement form. If A(7) = 0, the intermediate result is in true form. When the intermediate result is in complement form, the result must be recomplemented; that is, the 2's complement of the intermediate result is taken after the algebraic addition.

When the result is in true form, and if the fraction is not equal to zero, the fraction must be normalized and stored and the CC set. Because the AER instruction is a normalized instruction, the intermediate result is normalized, if necessary. After normalization, the sign and the characteristic are inserted and stored with the fraction

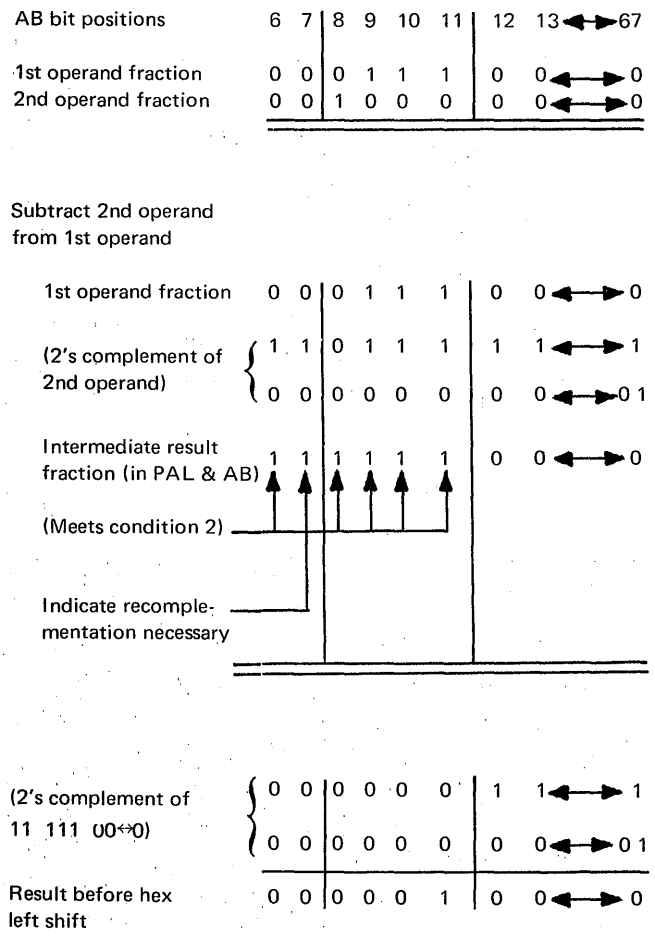
into the first operand location (specified by R1). Assuming no errors or zero fraction, the CC is set. An end-op cycle completes instruction execution.

When in the normalizing loop after subtraction, the intermediate fraction result can be left-shifted out of the high-order hex digit position if the intermediate fraction is 0001. This left shift results in a zero fraction. The zero fraction, in this case, is not a true zero result or a significance condition; therefore, the true value must be restored.

Because the test for ROS branches is made one machine cycle before the ROS branch occurs, a test for normalization is made before the recomplementation is performed. Therefore, the test for normalization is determined by the following conditions:

1. PAL(7–11) is 0's and PAL(7–67) is not 0's.
2. PAL(6,8–11) is 1's and PAL(7–67) is not 0's.

If one of these two conditions is met, at least one normalization cycle is performed after the recomplementation machine cycle. This action is not always necessary, however. For example, if the following fractions are subtracted, the assumed normalization cycle is not necessary:



The intermediate result fraction above shows that PAL(6,8–11) equals 1's and PAL(7–67) does not equal 0's. This condition causes a branch to the ROS normalization routine. Because A(7) = 1, the intermediate result fraction is in complement form, and the 2's complement of the intermediate fraction must be performed to obtain the true result fraction. As shown in the example, the true result fraction is .0001 0↔0. The one hex left shift that occurs yields a zero fraction result. Therefore, the result fraction located in DT is the true result fraction. The contents of D are transferred to T, and the sign, characteristic, and high-order fraction are stored into the FPR per the R1 field. An end op completes instruction execution.

If the signs were alike and the characteristic difference was within range (i.e., seven or less), the fractions are added [ROSAR(8–11) = 0110, 1110, or 1111]. First, however, the characteristics are equalized, if necessary, by right-shifting the fraction of the operand with the smallest characteristic one hex digit at a time and subtracting one from the characteristic difference. When the characteristic difference goes to zero, the characteristics are equal. Note that, if the first operand characteristic is larger, a 1 is added to the characteristic difference because the characteristic difference is in complement form.

The fractions are then added and the results are gated to DT and AB. A guard digit is retained in B(64–67). The characteristic of the larger operand, which is the characteristic of the intermediate result, is placed into F. When adding fractions, the possibility of a fraction overflow exists. A fraction overflow is indicated by a carry out of the high-order position, PA(8). After the two fractions are added, the intermediate result is placed into DT and AB. If A(7) = 1, a fraction overflow occurred. Therefore, the fraction is right-shifted one hex digit, and 1 is added to the intermediate result characteristic. Whether or not a fraction overflow occurred, the microprogram now determines whether the intermediate result fraction is normalized, not normalized, or equal to zero (the guard digit is included in the test for zero). If the fraction is normalized, the contents of STAT C are gated to the sign position of the LS bus, the result characteristic is transferred from F to T(32–39), and the result fraction is gated from D to T(40–63). This result (sign, characteristic, and fraction) is stored into the FPR specified by the R1 field. If the intermediate result fraction is not normalized, it is normalized by shifting left one hex digit at a time and subtracting one from the characteristic until a significant hex digit appears in the high-order position. After normalizing, the operation continues in the same manner as a normalized intermediate fraction. If the intermediate result is zero, a true zero is stored into the FPR specified by the R1 field, and the significance mask bit [PSW(39)] is tested. If the mask bit is a 1, the characteristic is stored with the zero fraction, and a program interruption is

initiated. Regardless of the setting of the mask bit, the CC is set to 0, and the instruction is terminated by an end op.

If the intermediate result fraction is shifted right, the possibility of an exponent overflow exists. During normalization of the fraction, the possibility of an exponent underflow exists. When SAL(0) = 1 after a fraction shift (right or left), an exponent overflow or exponent underflow condition exists. The proper sign and fraction are stored. Because bit 0 of the characteristic (characteristic carry) is inhibited from entering LS by the 'RSLT-SIGN→LS' micro-order, the characteristic stored is 128 larger than the true result for underflow and 128 smaller on overflow.

Whenever an exponent overflow or exponent underflow condition exists, SAL(1) and PSW(38) are examined to determine whether a program interruption is to be executed. If SAL(1) = 0, an exponent overflow exists, and an interruption request is unconditionally generated. A program interruption occurs on all exponent overflows. If SAL(1) = 1, an exponent underflow exists; if PSW(38) = 1, an exponent underflow program interruption request is generated. If PSW(38) = 0, exponent underflow is masked off, and a true zero is stored with no program interruption occurring.

Note that an interruption occurs on all exponent overflows. This overflow indicates that the value of the absolute result exceeds the limits of the machine; therefore, further action is necessary. In some scientific computations, very small numbers may be eliminated from an equation without serious error. An exponent underflow means that the computed result approaches zero. Therefore, the programmer may find that a program interruption is unnecessary, and a true zero result is desirable.

Significance and specification program interruption conditions may also exist during execution of an AER instruction. The action that occurs is shown in Diagram 5-206, and is discussed earlier in this section.

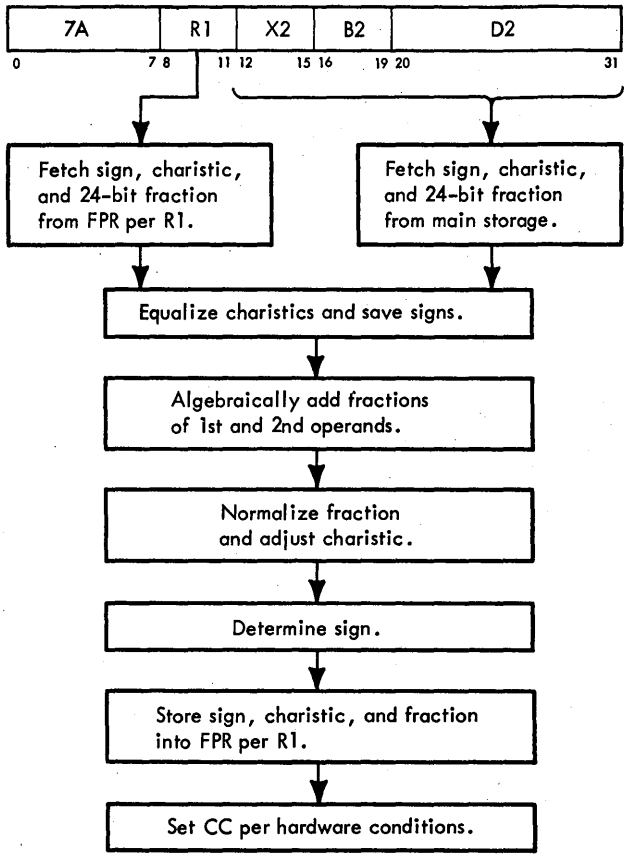
Tests for zero intermediate results are made at several points during instruction execution. If the result is zero, a program interruption occurs if PSW(39) = 1. The positive sign, the result characteristic, and a zero fraction are stored into LS. A program interruption occurs, and the program interruption routine determines the action to be taken. If PSW(39) = 0, a true zero result is stored into LS.

If the characteristics are not within limits when executing the AER instruction [ROSAR(8–11) = 0000, 0100, 1000, or 1100 on 'FLR' branch], the fraction with the largest characteristic is normalized and stored along with the sign into LS per the R1 field.

This discussion of the AER instruction is referred to in the discussions of Add-type instructions that follow. If the AER instruction is understood and the instruction differences noted, any short operand add-type instruction execution path can be followed by referring to Diagram 5-206.

Add Normalized, AE (7A) – RX Short Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1) and place normalized sum into 1st operand location.
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AE, instruction (Diagram 5-206) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AE instruction uses 32-bit operands; therefore the contents of the low-order halves of the FPR's remain unchanged.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.

2. A main storage request for the second operand has been issued per the effective address in D.
3. The contents of A and B are unknown.
4. The first 16 bits of the instruction are in E.

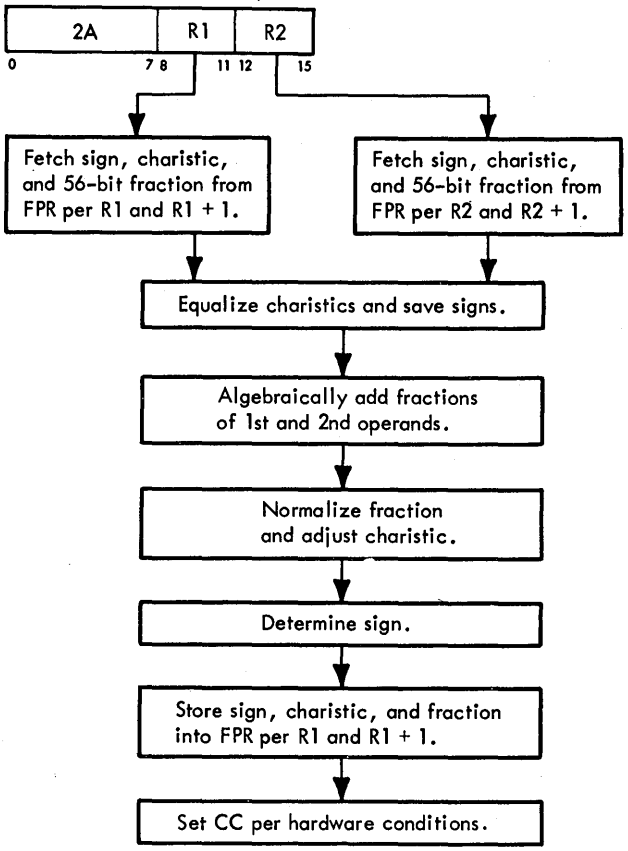
A specification test is made at the beginning of the execution phase. If a specification check exists, instruction execution is suppressed, and a specification program interruption occurs. Assume that no specification check exists.

Because the AE instruction uses short operands (32 bits) in the RX format, no low-order fractions need to be fetched. The first operand is moved from T to A. The second operand arrives from main storage and is placed into T. D(21) determines which word from the SDBO is gated to T. (Note that main storage is addressed on doubleword boundaries.) If D(21) = 0, SDBO(0–31) is gated to T; if D(21) = 1, SDBO(32–63) is gated to T.

The remainder of the AE instruction execution is identical to the execution of the AER instruction.

Add Normalized, ADR (2A) – RR Long Operands

- Algebraically add 2nd operand (in FPR, per R2 and R2 + 1) to 1st operand (in FPR, per R1 and R1 + 1) and place normalized sum into 1st operand location.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, ADR, instruction (Diagram 5-207, FEMDM) algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The ADR instruction uses 64-bit operands.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in A and B, and D contains the high-order 24 bits of the fraction.
2. 32 bits of the second operand (sign, characteristic, and high-order fraction) are in S and T.
3. The STC contains a count of 4.
4. The instruction is in E.

Because the ADR instruction uses long operands (64 bits) in the RR format, the low-order fractions of the first and second operands must be fetched from LS. The low-order fraction of the first operand is fetched from LS per E(8–11) + 1 and is placed into B via T and the parallel adder. The low-order fraction of the second operand is fetched from LS per E(12–15) + 1 and placed into T; the high-order fraction is placed into D.

The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation by means of a 10-way 'FLR' branch.

Note that with long operands, the characteristics can be equalized if the characteristic difference is less than or equal to 15. Assume that the following two characteristics are to be compared to determine the ROSAR value:

2nd operand characteristic: $64_{10} = 1000000_2$

1st operand characteristic: $73_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic as follows:

2nd operand characteristic:	0 1000000	
Complement of 1st operand characteristic:	1 0110110	2's com-
Add 1:	0 0000001	plement
Characteristic difference in SAL(0–7):	1 1110111	

Because no SAL(0) carry occurred, ROSAR(8) = 0. ROSAR(9) depends upon the signs assigned to the fractions of the two operands. The table in Sheet 2 of Diagram 5-207 shows the bit positions tested in SAL to determine the value of ROSAR(10) and whether preshifting is meaningful. In this example, SAL(0–3) = 1111. Therefore, because ADR is a long operand instruction, ROSAR(10) is set to 1 and preshifting is meaningful. ROSAR(11) = 0 because SAL(0–7) is not all 0's. For other examples of ROSAR setting on an 'FLR' branch, refer to the description of the AER instruction and Table 3-9. The examples shown in Table 3-9 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is 15 or less for long operand instructions.

Four possible ROSAR(8–11) values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition or subtraction of fractions. For example, ROSAR(8–11) is set to 0110, if an ADR instruction is being executed with operands having the following parameters: (1) the characteristic difference is less than 15, (2) the signs of the operands are alike, and (3) the second operand characteristic is the smaller of the two. The 0110 branch right-shifts the second operand in DT one hex digit and adds 1 to the characteristic difference in F until the characteristics are equal [SAL(4–7) = 1111].

After the characteristics are equalized, the fractions of the operands are added and the sum is placed into AB and DT. A plus sign is set into F(0) (in case of a zero fraction result), and the characteristic of the first operand is placed into F(1–7). STAT A is set if the fraction of the result equals zero.

Because ADR is a normalized instruction, the intermediate result is normalized, if necessary. First, however, a test determines whether the fraction overflowed due to the addition. (A carry into bit 7, the low-order bit of the characteristic, is considered a fraction overflow.) If the fraction did overflow, it is shifted right one hex digit, and a 1 is added to the characteristic.

The low-order fraction is then stored into the FPR specified by R1 + 1, and normalization proceeds if the result was not zero and not normalized. The low-order fraction is stored after each left-shift.

After normalization, the sign and the characteristic are inserted and stored with the high-order fraction into the first operand location (specified by R1). Assuming no error conditions or zero fraction, the CC is set, and an end-op cycle completes the execution.

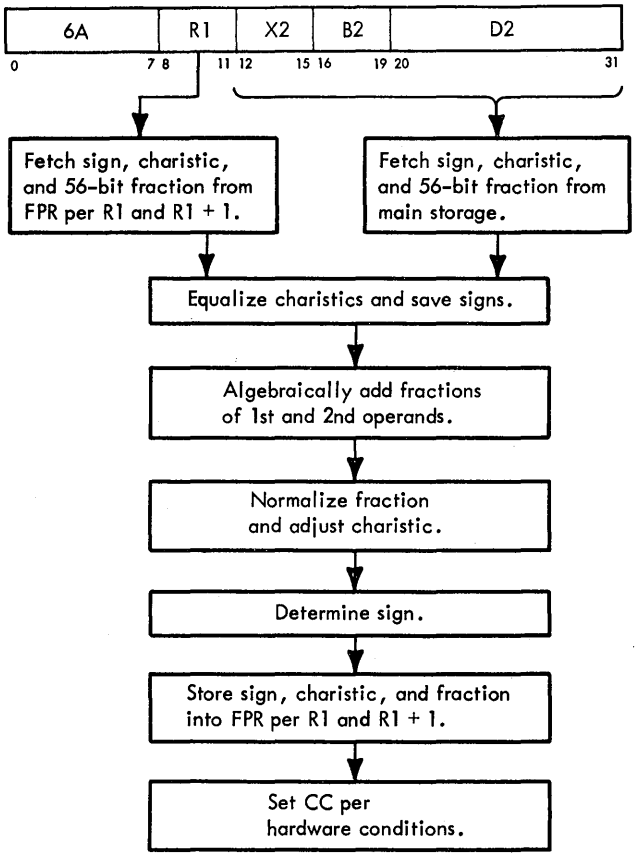
If signs are unlike, when in the normalizing loop, the intermediate result fraction can be left-shifted out of the high-order hex digit position if the intermediate fraction is 0001. Refer to the description of the AER instruction for an explanation of how this problem is handled.

All other operations that may occur during the execution of an ADR instruction are handled as described for an AER instruction, with the exception that the low-order fraction must be considered in all calculations. The major differences are:

1. An additional operand fetch is needed.
2. The low-order halves of the FPR's are used.

Add Normalized, AD (6A) – RX Long Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1 and R1 + 1) and place normalized sum into 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

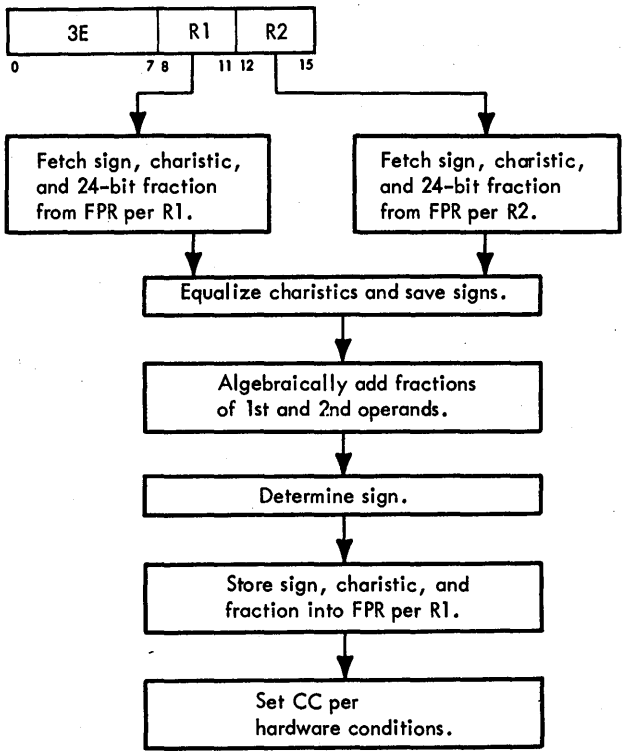
The Add Normalized, AD, instruction (Diagram 5-207) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1 and R1 + 1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AD instruction uses 64-bit operands.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The contents of A and B are unknown.
4. The first 16 bits of the instruction are in E.

Because the AD instruction uses long operands (64 bits) in the RX format, the low-order fraction of the first operand must be fetched from LS. Accordingly, the sign, characteristic, and high-order fraction of the first operand are placed into A. The low-order fraction is then fetched from LS per E(8–11) and routed to B via T and the parallel adder. The 64-bit second operand is fetched from main storage per D and placed into ST; the high-order fraction is also placed into D. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C.

The first operand characteristic is then subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation by means of a 10-way 'FLR' branch. The remainder of the execution of the AD instruction is identical to that of the ADR instruction.



Add Unnormalized, AUR (3E) – RR Short Operands

- Algebraically add 2nd operand (in FPR, per R2) to 1st operand (in FPR, per R1) and place unnormalized sum into 1st operand location.
- RR format: (Shown in adjacent column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

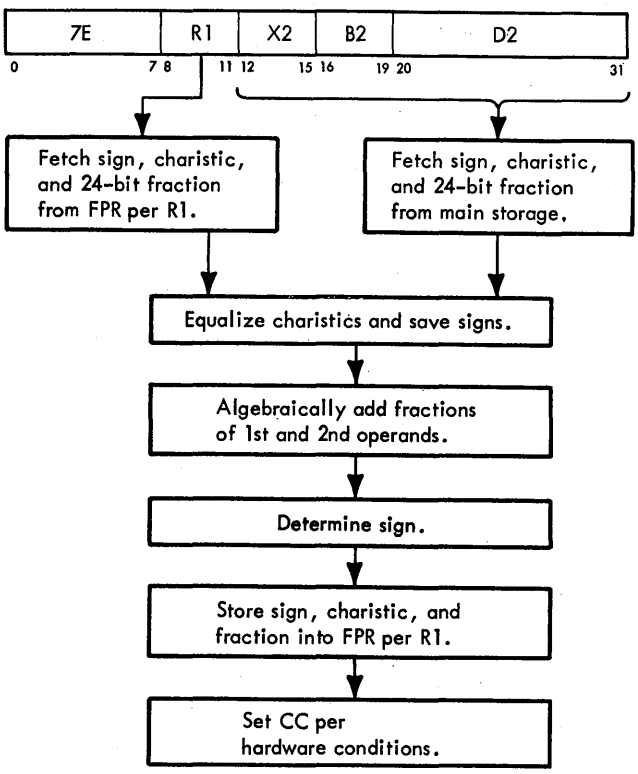
The Add Unnormalized, AUR, instruction (Diagram 5-206) algebraically adds the second operand (specified by R2) to the first operand (specified by R1), and places the unnormalized sum into the first operand location. The CC is set according to hardware conditions. Note that the guard digit is not examined to determine the CC setting or checked for a significance condition. Also, because the characteristic is not reduced for normalization, exponent underflow cannot occur.

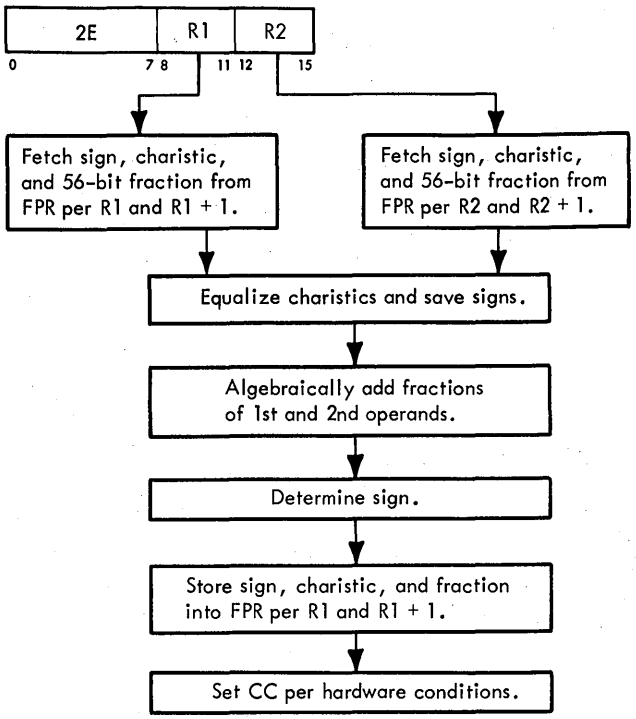
The execution of the AUR instruction is identical to that of the AER instruction until the branch on the type of instruction is performed. After the fraction is tested for overflow and shifted right (if necessary), the result is checked to see if it is zero. If the result is not zero, the result is immediately stored, along with the sign and characteristic, into the FPR specified by R1.

A test then determines whether equalization had occurred (STAT D set). If it did, a possibility exists that the only significant digit may be the guard digit. Therefore, if equalization did occur and no exponent overflow condition exists, the guard digit is removed from the result and the fraction is tested to see if it is zero. If the fraction equals zero, a true zero is stored and a significance program interruption is initiated. Otherwise, the instruction terminates normally.

Add Unnormalized, AU (7E) – RX Short Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1) and place unnormalized sum into 1st operand location.
- RX format: (See left column of next page.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.





- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AU, instruction (Diagram 5-206) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1), and places the unnormalized sum into the first operand location. Exponent underflow cannot occur. The CC is set according to hardware conditions. The execution is identical to that of the AE instruction until the branch to determine an unnormalized, normalized, or compare instruction. From this point, the execution is identical to that of the AUR instruction.

Add Unnormalized, AWR (2E) – RR Long Operands

- Algebraically add 2nd operand (in FPR, per R2 and R2 + 1) to 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized sum into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

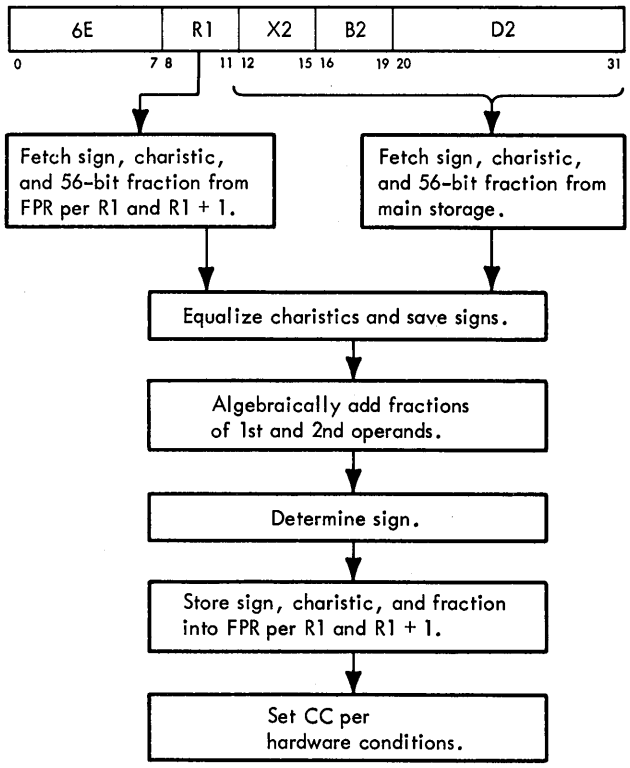
The Add Unnormalized, AWR, instruction (Diagram 5-207) algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1) and places the unnormalized sum into the first operand location. Exponent underflow cannot occur.

The execution phase is identical to that of the ADR instruction until the branch to determine an unnormalized, normalized, or compare instruction. Assume that the operand signs are alike and that the characteristic difference is less than 15. On Sheet 3 of Diagram 5-207, after the branch, a test determines whether the fraction of the sum overflowed (indicated by a carry into bit 7). If an overflow occurred, the fraction is shifted right one digit, and a 1 is added to the characteristic.

The low-order position of the sum (bits 32–63) is then stored into the FPR designated by R1 + 1. The result fraction is then tested to see if it equals zero. If it does not equal zero, the sign, characteristic, and high-order fraction are stored into the FPR specified by R1. If no exponent overflow occurred, the CC is set and the instruction is terminated by an end op.

Add Unnormalized, AW (6E) – RX Long Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized sum into 1st operand location.
- RX format:



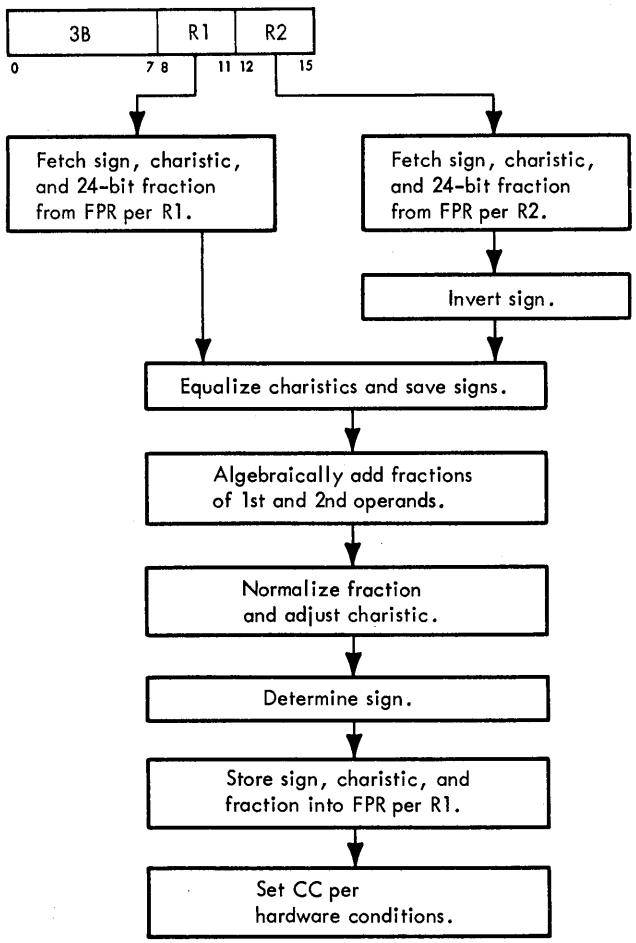
- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AW, instruction (Diagram 5-207) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1 and R1 + 1) and places the unnormalized sum into the first operand location. Exponent underflow cannot occur. The CC is set according to hardware conditions.

The execution phase is identical to that of the AD instruction until the branch to determine an unnormalized, normalized, or compare instruction. From this point, execution is identical to execution of the AWR instruction.

Subtract Normalized, SER (3B) – RR Short Operands

- Algebraically subtract 2nd operand (in FPR per R2) from 1st operand (in FPR, per R1) and place normalized difference into 1st operand location.
- RR format:



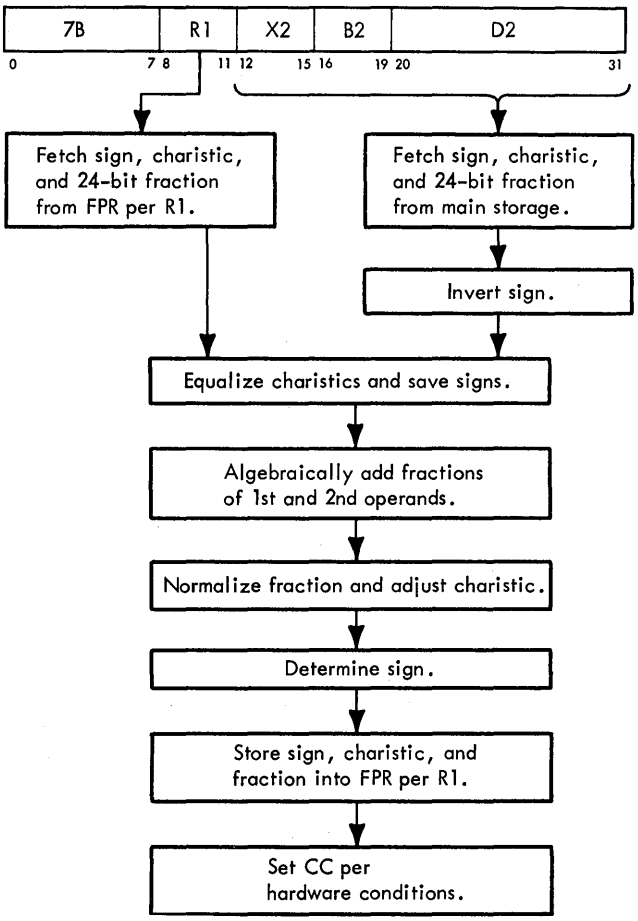
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SER, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by R2) from the first operand (specified by R1), and places the normalized difference into the first operand location. The second operand location remains unchanged. The CC is set according to hardware conditions.

For subtract instructions, the sign of the second operand is complemented, after which the algebraic subtraction is treated as an algebraic addition. Therefore, execution of the SER instruction is identical to that of the AER instruction except that the complement of the second operand sign rather than the true sign is gated to STAT C.

Subtract Normalized, SE (7B) – RX Short Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1) and place normalized difference into 1st operand location.
- RX format:



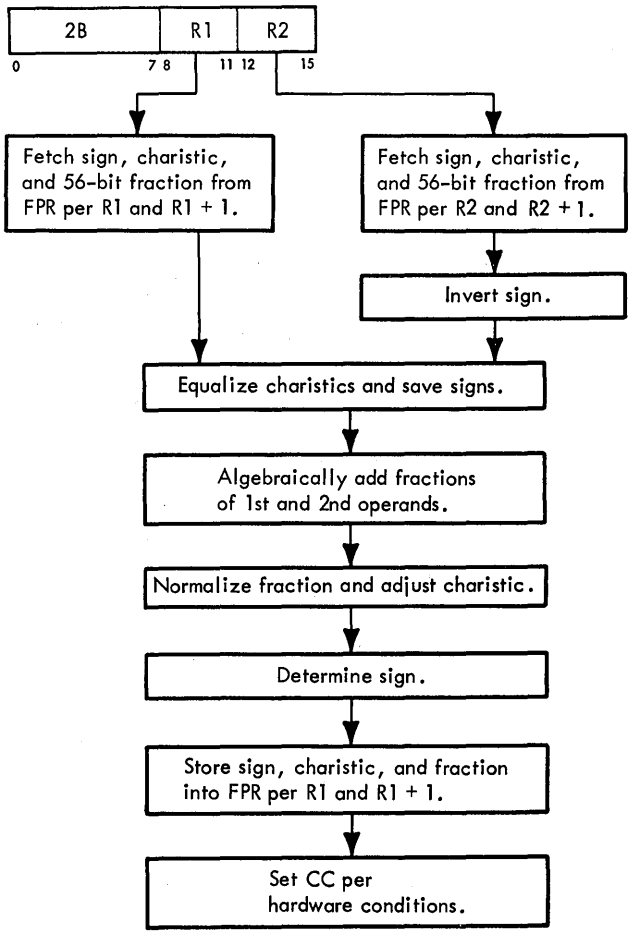
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SE, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SE instruction is identical to that of the AE instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Normalized, SDR (2B) – RR Long Operands

- Algebraically subtract 2nd operand (in FPR, per R2 and R2 + 1) from 1st operand (in FPR, per R1 and R1 + 1) and place normalized difference into 1st operand location.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

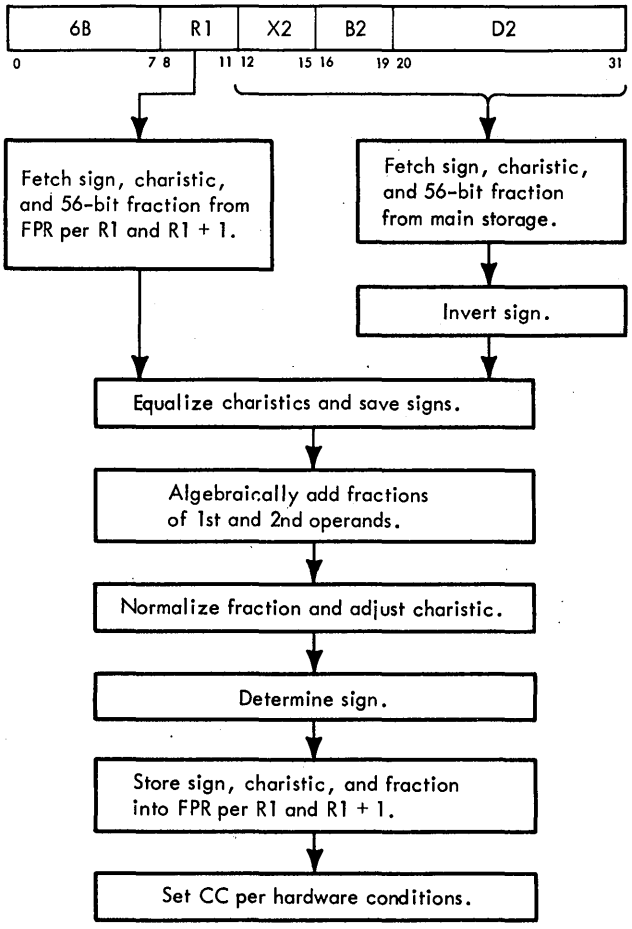
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SDR, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by R2 and R2 + 1) from the first operand (specified by R1 and R1 + 1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SDR instruction is identical to that of the ADR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Normalized, SD (6B) – RX Long Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1 and R1 + 1) and place normalized difference into 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

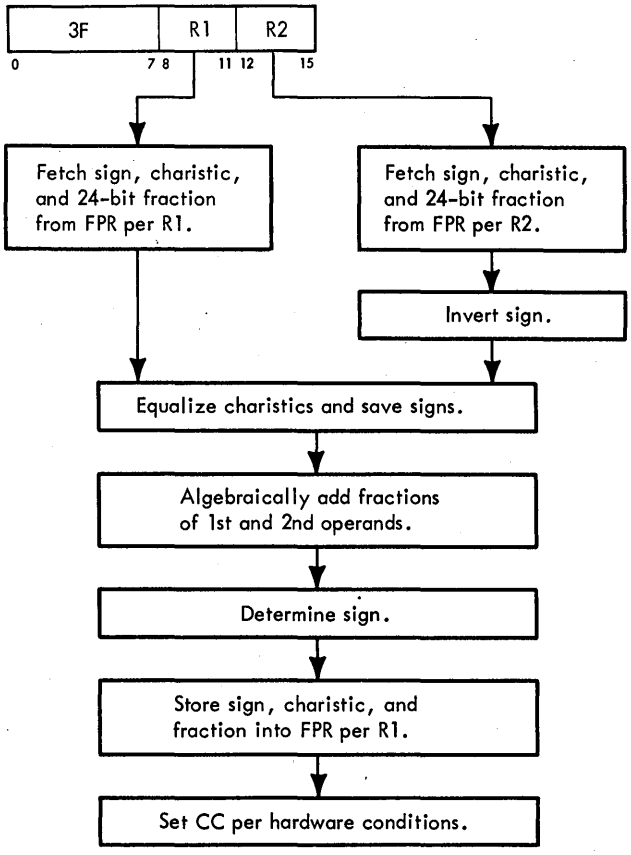
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SD, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1 and R1 + 1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SD instruction is identical to the execution of the AD instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SUR (3F) – RR Short Operands

- Algebraically subtract 2nd operand (in FPR, per R2) from 1st operand (in FPR, per R1) and place unnormalized difference into 1st operand location.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SUR, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by R2) from the first operand (specified by R1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Except that the intermediate results of the Subtract Unnormalized instructions are not normalized, the operation is the same as that of the Subtract Normalized instructions. That is, the second operand sign is inverted (and saved in STAT C) and the fractions are algebraically added.

The SUR instruction is executed in the same manner as the AUR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C. Note that, when executing Subtract Unnormalized instructions, the guard digit is not examined to determine the CC setting or checked for a significance condition. Also, as in Unnormalized Add instructions, exponent underflow cannot occur.

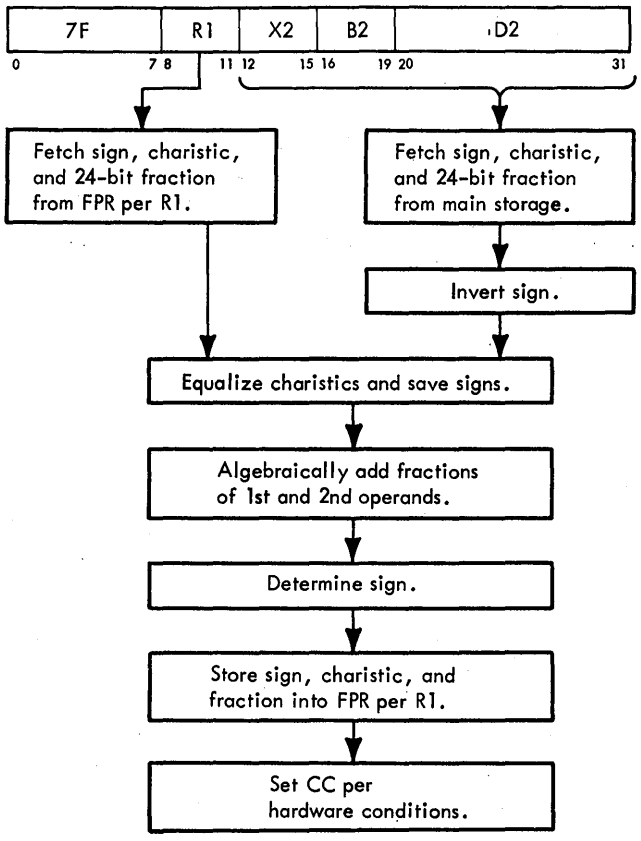
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SU, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SU instruction is identical to that of the AU instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

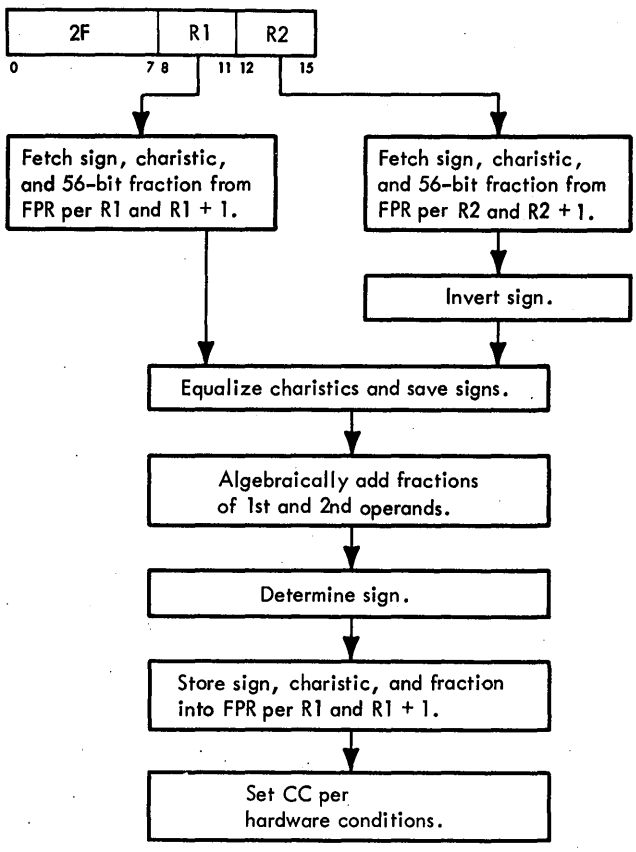
Subtract Unnormalized, SU (7F) – RX Short Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1) and place unnormalized difference into 1st operand location.
- RX format:



Subtract Unnormalized, SWR (2F) – RR Long Operands

- Algebraically subtract 2nd operand (in FPR, per R2 and R2 + 1) from 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized difference into 1st operand location.
- RR format:



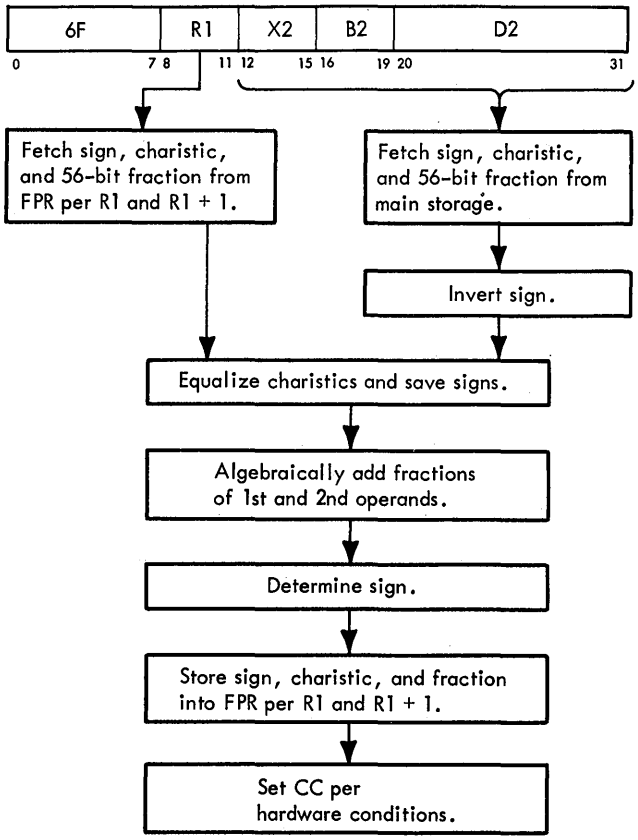
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SWR, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by R2 and R2 + 1) from the first operand (specified by R1 and R1 + 1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SWR instruction is identical to that of the AWR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SW (6F) – RX Long Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized difference into 1st operand location.
- RX format:



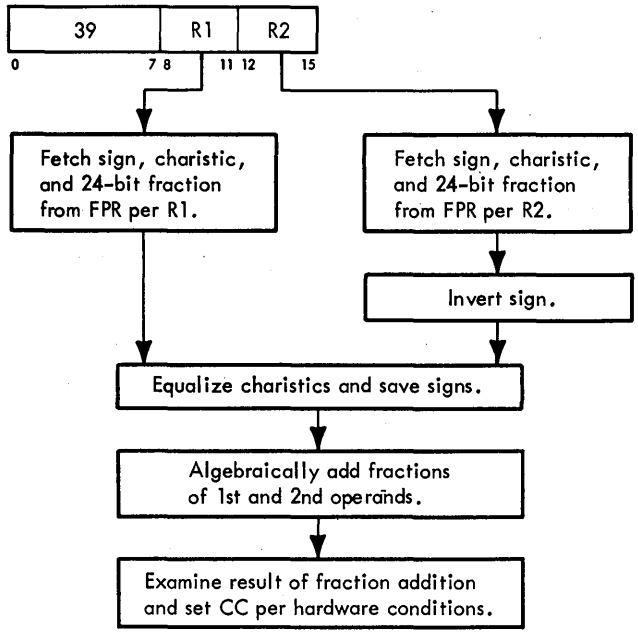
- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SW, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1 and R1 + 1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SW instruction is identical to that of the AW instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Compare, CER (39) – RR Short Operands

- Algebraically compare 1st operand (in FPR, per R1) with 2nd operand (in FPR, per R2); CC indicates result.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CER, instruction (Diagram 5-206) algebraically compares the first operand (specified by R1) with the second operand (specified by R2); the CC indicates that the first operand is equal to, less than, or greater than the second operand.

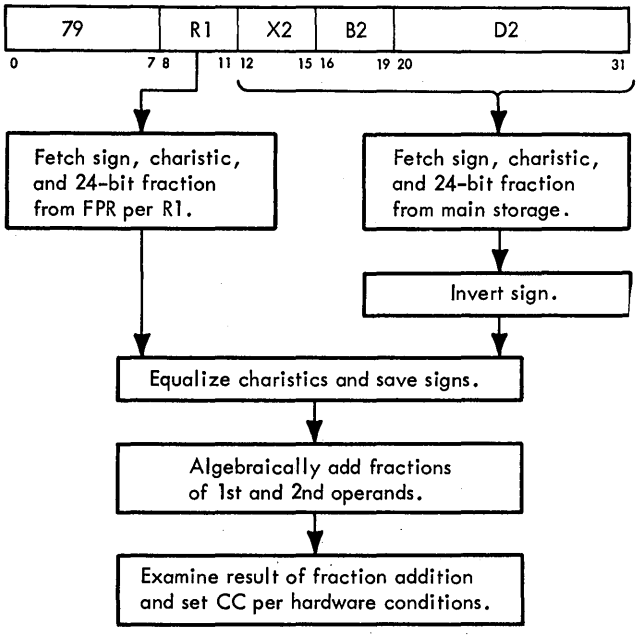
Comparison is algebraic, taking into account the sign, fraction, and characteristic of each operand. An exponent inequality is not decisive for magnitude determination because the fractions may have different numbers of leading zeros. Equality is established by following the rules for floating-point subtraction. The intermediate result is not normalized or stored. When the intermediate result, including a possible guard digit, is zero, the operands are equal. Numbers with zero fractions compare equal even when they differ in sign or characteristic. Exponent overflow, exponent underflow, or significance check cannot occur. The CC is set per hardware conditions at end-op time.

In the CER instruction, the contents of the low-order halves of the FPR's are ignored. (Neither operand location is changed as a result of any compare instructions.)

Execution of the CER instruction is identical to that of the SER instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CE (79) – RX Short Operands

- Algebraically compare 1st operand (in FPR, per R1) with 2nd operand (in storage); CC indicates result.
- RX format:



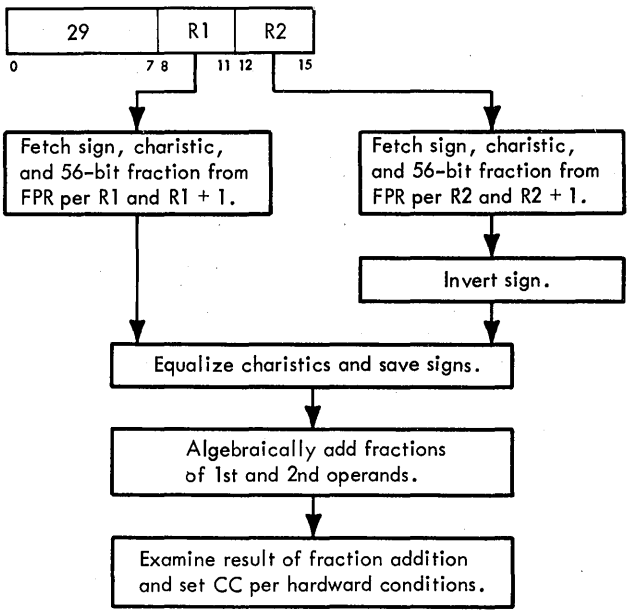
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  1st 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CE, instruction (Diagram 5-206) algebraically compares the first operand (specified by R1) with the second operand (specified by the effective address); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CE instruction is identical to that of the SE instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CDR (29) – RR Long Operands

- Algebraically compare 1st operand (in FPR, per R1 and R1 + 1) with 2nd operand (in FPR, per R2 and R2 + 1); CC indicates result.
- RR format:



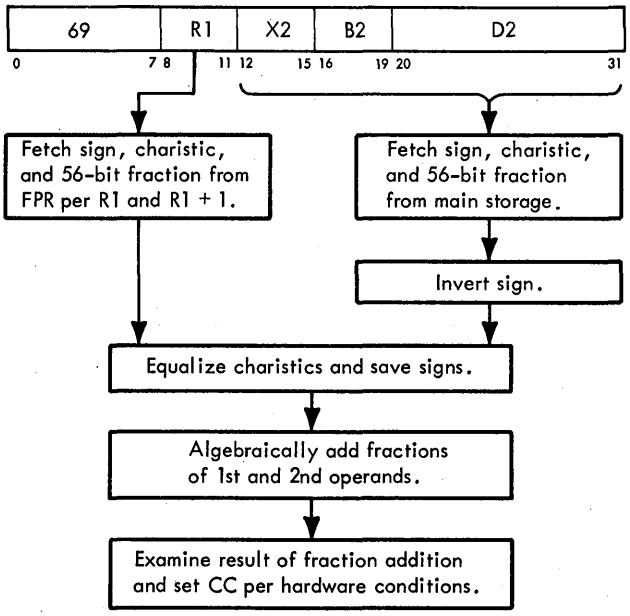
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CDR, instruction (Diagram 5-207) algebraically compares the first operand (specified by R1 and R1 + 1) with the second operand (specified by R2 and R2 + 1); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CDR instruction is identical to that of the SDR instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CD (69) – RX Long Operands

- Algebraically compare 1st operand (in FPR, per R1 and R1 + 1) with 2nd operand (in storage); CC indicates result.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CD, instruction (Diagram 5-207) algebraically compares the first operand (specified by R1 and R1 + 1) with the second operand (specified by the effective address); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CD instruction is identical to that of the SD instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

HALVE

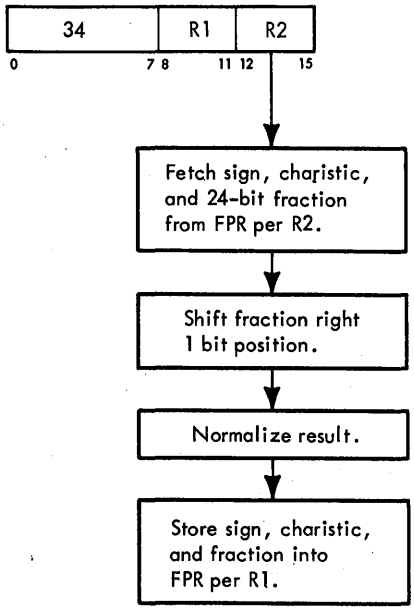
The Halve instructions (HER and HDR) divide the second operand by 2 and place the normalized quotient into the first operand location. The Halve instructions are in the RR format with short and long operand options available. In the HER instruction, the low-order half of the result register remains unchanged.

After the second operand is in ST, the sign and the characteristic are saved in F, and the high-order fraction (24 bits) is placed into D (long operands). Shifting the fraction one bit position to the right divides the operand by 2. Because the data in PAL cannot be shifted right 1 directly, two machine cycles are necessary. First the fraction is shifted left 1 from D to the parallel adder and then shifted right 4 to PAL, thus yielding an effective right 3 shift. Next, the fraction is placed into AB. A left 2 shift occurs when the fraction is routed to DT via the parallel adder, thus resulting in a right 1 shift and thereby dividing the fraction by 2. Guard digits are saved and used if normalization is necessary. After the fraction is normalized, the sign, characteristic, and fraction are stored into LS per R1, completing instruction execution.

The halve operation differs from the divide operation in that 2 is the only divisor.

Halve, HER (34) – RR Short Operands

- Divide 2nd operand (in FPR, per R2) by 2 and place normalized quotient into 1st operand location (in FPR, per R1).
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Halve, HER, instruction (Diagram 5-208, FEMDM) divides the second operand (specified by R2) by 2 and places the normalized quotient into the FPR specified by R1. To divide by 2, the fraction is shifted right one bit position.

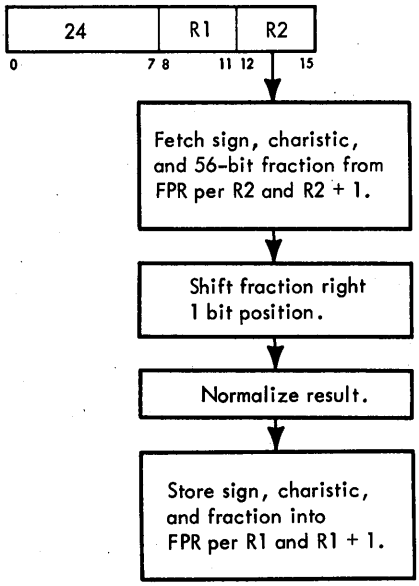
At the start of execution, the second operand is in S and T. After a specification test, the fraction is placed into D, the sign is saved in STAT C, and the characteristic is saved in F. The fraction in D is then shifted right one bit position. A test is initiated to determine whether the fraction is zero and normalized. If the fraction equals zero, the FPR specified by R1 is set to a true zero and the operation terminates with an end op. If the fraction contains leading zero's, it is normalized by shifting left one hex digit at a time until a significant hex digit appears in the high-order position. A 1 is subtracted from the characteristic for each shift.

After normalization, the characteristic and fraction are gated to T from F and D, respectively. The result is then stored into the FPR specified by R1. The sign position is set by forcing the contents of STAT C onto the LS bus as the fraction and characteristic are being stored.

The characteristic is then tested for a possible exponent underflow. If no underflow occurred, the instruction is terminated. If an underflow did occur, the remainder of the operation is determined by the state of the underflow mask bit.

Halve, HDR (24) – RR Long Operands

- Divide 2nd operand (in FPR, per R2 and R2 + 1) by 2 and place normalized quotient into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Halve, HDR, instruction (Diagram 5-209, FEMDM) divides the second operand, (specified by R2 and R2 + 1) by 2 and places the normalized quotient into the FPR specified by R1 and R1 + 1. To divide by 2, the fraction is shifted right one bit position.

At the start of the execution phase, the sign, characteristic, and high-order fraction are in S and T. After a specification test, the high-order fraction is placed into D and the low-order fraction is fetched from LS per R1 + 1 and gated to T. The sign is saved in STAT C, and the characteristic is saved in F. The fraction in DT is then shifted right one bit position, and a test determines whether the fraction is equal to zero and normalized. If the fraction equals zero, the FPR specified by R1 and R1 + 1 is set to a true zero and the operation terminates with an end op. If the fraction contains leading zero's, it is normalized by shifting left one hex digit at a time until a significant hex digit appears in the high-order position. A 1 is subtracted from the characteristic for each shift.

After normalization, the low-order fraction is stored into the FPR specified by R1 + 1. The characteristic and high-order fraction are then gated to T from F and D, respectively. The result is stored into the FPR specified by R1. The sign position is set by forcing the contents of STAT C onto the LS bus as the fraction and characteristic are being stored.

The characteristic is then tested for a possible exponent underflow. If no underflow occurred, the instruction is terminated. If an underflow did occur, the remainder of the operation is determined by the state of the underflow mask bit.

MULTIPLY

- Multiplies 1st operand and 2nd operand and places normalized product into 1st operand location.
- Note: In CE floating-point multiply operations, 2nd operand is multiplicand and 1st operand is multiplier.
- Product is 64 bits for both short and long operand instructions.
- Characteristics are added and 64 is subtracted to obtain intermediate characteristic.
- Operands are prenormalized before multiplying.

- Product is normalized before storing.
- Sign of product is determined algebraically.

Two floating-point numbers are multiplied by adding their characteristics and multiplying their fractions. For example, if 1250 is to be multiplied by 5, converting these numbers to hex notation yields the equation

$$(.4E2_{16} \times 16^3) \times (.6 \times 16^1).$$

The product is obtained, as follows:

$$.4E2_{16} \times 16^3$$

$$\times \quad .5_{16} \times 16^1$$

Product of fractions → $.186A_{16} \times 16^4$ ← Exponents are added

The product ($.186A_{16} \times 16^4$) equals 6250 (decimal), which is the product of 1250 x 5.

When two floating-point numbers are multiplied, the characteristics are added to yield the final characteristic value of the product, as shown above. Because excess-64 notation is used, 64 must be subtracted from the characteristic sum because the characteristic value is in excess-128 [(C1 + 64) + (C2 + 64) = C1 + C2 + 128] after characteristic addition. When 64 is subtracted, the result is returned to excess-64 notation (C1 + C2 + 128 - 64 = C1 + C2 + 64). For instance, in floating-point format, the exponents used in the example above yield characteristics of 67 (3 + 64) and 65 (1 + 64). The sum of these characteristics is 132. Subtracting 64 from 132 leaves 68, which is equivalent to an exponent of 4 in excess-64 notation.

If one or both operand fractions contain leading zeros, the unnormalized operand(s) is prenormalized. That is, the operands are normalized before multiplication begins. Prenormalization increases product precision. By prenormalizing the operands, a maximum of one postnormalization cycle is necessary. Postnormalization is the process of normalizing the product after fraction multiplication. Prenormalization and postnormalization are accomplished by shifting the fraction left one hex digit and subtracting 1 from the characteristic value for each left shift until a significant hex digit appears in the high-order position of the fraction.

The product for both short and long operand multiply instructions is 64 bits long. Note that, if the fraction is not prenormalized, dropping the low-order bits of the product in excess of 64 may result in a false zero product without prenormalization. This result would occur often in long operand instructions because 56 low-order bits of the product are lost when executing the multiply algorithm. Thus, to prevent a false zero, the product for long operand instructions would have to be 120 bits long. A false zero is prevented, however, by prenormalizing the operands and postnormalizing the intermediate product. During postnormalization, the intermediate product characteristic is reduced by the number of left shifts. For long operands, the low-order bits of the intermediate product are dropped before left-shifting. For short operands (six-digit fractions), the product fraction has the full 14 hex digits of the long format, and the two low-order hex fraction digits are accordingly always zeros. The two low-order hex fraction digits are zeros in short operand instructions because a maximum of 12 nonzero hex digits is possible when multiplying two six-digit numbers. In multiplication, the number of digits in the product cannot exceed the sum of the available operand digits. Therefore, because 14 product digits are available, the two low-order hex digits of short operand products are always zeros.

The sign of the product is determined algebraically; that is, if the signs of the operands are alike (both plus or both minus), the product is assigned a plus sign; if the signs are unlike, the product is made negative.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is terminated, and a program interruption occurs. The overflow interruption condition does not occur for a partial product characteristic exceeding 127 when the final characteristic is brought within range through normalization.

When exponent underflow occurs, the final product characteristic is less than zero. The sign, characteristic, and fraction are made zero, and a program interruption occurs if the corresponding mask bit is a 1. Underflow is not signalled when the characteristic of an operand becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 14 result fraction digits are zero, the product sign and characteristic are made zero, yielding a true zero result, exponent underflow is not signalled, and no interruption is taken. The program interruption for significance is never taken for multiplication.

Data Flow and Algorithm

- See Note under "Multiply".
- Signs are saved in STAT's C and F.
- Characteristics are added in serial adder; carry is saved in STAT D.
- Fraction multiplication is performed by multiply/divide logic.
- E(12–15) selects two multiplier bits from S.
- S bits determine multiple value to be added to partial product.

The data paths of the floating-point operands during multiplication are shown in Diagram 5-210, FEMDM. The characteristic is computed in the serial adder (A of the

diagram). The signs are saved in STAT C and STAT F. To add the characteristics, the first operand characteristic is gated to SAA(1–7) from AB per the ABC, and the second is gated to SAB(1–7) from ST per the STC. The characteristic sum is routed to F and the characteristic carry is saved in STAT D and F(0). 64 is subtracted from the characteristic by adding the 2's complement of 64 to the sum in F.

Note: For an RX instruction with a normalized first operand, the first operand characteristic and sign are in S or T and the second operand characteristic and sign are in A.

Fraction multiplication is performed by the multiply/divide logic (B of Diagram 5-210) and is similar to fixed-point multiplication (Section 2 of this Chapter) except that the operands are shorter. After the signs are saved, the characteristic is determined and the operands are prenormalized; the multiplicand is in DT, the multiplier is in S, and a count, representing the number of repetitive operations necessary to perform the multiplication, is in E(12–15). Multiplication is performed two bits at a time; that is, the multiplicand in DT is multiplied by two bits of the multiplier in S using the multiple-selection decoder and the parallel adder. The count in E(12–15), in addition to keeping track of the number of operations, determines which multiplier bits are to be used, starting with the low-order bits and moving two bits to the left for each multiplication. The result of each two-bit multiplication is a multiple of the multiplicand which is then added to the partial product formed by previous two-bit multiplications. Thus, a new partial product is obtained. The two-bit multiplications continue until E(12–15) indicates that all bits of the multiplier have been used. At that time, the intermediate product is contained in AB(4–67).

The steps of the fraction multiplication are:

1. Place the constant F (hex) into E(12–15).
2. Using the value in E(12–15), select two multiplier bits (M1 and M2) from S. E(12,13) selects the byte and E(14,15) selects the two bits within a byte (see B of Diagram 5-210). For example, with the original value in E(12–15), E(12,13) = 11 selecting the third byte in S (bits 24–31), and E(14,15) = 11 selecting bits 6 and 7 within that byte. Thus, the first M1, M2 values used are S(30,31). Table 3-10 shows which S bits are selected for all values of E(12–15).
3. The value of the M1, M2 bits, in conjunction with the 'TX' trigger, gates the correct multiple of the multiplicand to the PAA. Considering the M1, M2 bits as a two-bit multiplier, the multiplicand in DT can be multiplied by 0 (M1, M2 = 00), by 1 (M1, M2 = 01), by 2 (M1, M2 = 10), or by 3 (M1, M2 = 11), as follows:
   a. M1, M2 = 00 and 'TX' Trigger Is Reset: Because zero times the multiplicand is zero, nothing is added to the partial product. However, the partial product is shifted right two bit positions.
   b. M1, M2 = 01 and 'TX' Trigger Is Reset: One times the multiplicand equals the multiplicand. Thus, the multiplicand is added to the partial product, and the partial product is shifted right two bit positions.
   c. M1, M2 = 10 and 'TX' Trigger Is Reset: The multiplicand is multiplied by 2 by shifting it left one bit position. The result is then added to the partial product, and the partial product is shifted right two bit positions.
   d. M1, M2 = 11 and 'TX' Trigger Is Reset: Because the facilities for multiplying the multiplicand by 3 are not directly available, an effective multiplication by 3 is accomplished by multiplying by 4 and subtracting one times the multiplicand (4X - 1X = 3X). The minus 1X occurs first by adding the 2's complement of the multiplicand to the partial product, and 4X occurs by adding 1 to the next two multiplier bits. The 'TX' trigger is set to remember that an additional 1 is required in the next cycle. After adding the 2's complement of the multiplicand to the partial product, the partial product is shifted right two bit positions.
   e. M1, M2 = 00 and 'TX' Trigger Is Set: One times the multiplicand is added to the partial product, which is then shifted right two bit positions.
   f. M1, M2 = 01 and 'TX' Trigger Is Set: Two times (left 1 shift) the multiplicand is added to the partial product, which is then shifted right two bit positions.
   g. M1, M2 = 10 and 'TX' Trigger Is Set: The 2's complement of the multiplicand is added to the partial product, and the 'TX' trigger is again set. The resulting partial product is shifted right two bit positions.
   h. M1, M2 = 11 and 'TX' Trigger is Set: Zero is added to the partial product, and the 'TX' trigger is set. The partial product is shifted right two bit positions.

Note that in each case the new partial product is shifted right two bit positions. Thus, each higher-order multiplicand multiple is displaced two positions to the

Table 3-10. Multiplier Bits Selected, Floating-Point Multiply

	E(14,15)							
	00		01		10		11	
E(12,13)	M1	M2	M1	M2	M1	M2	M1	M2
00	0	1	2	3	4	5	6	7
01	8	9	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23
11	24	25	26	27	28	29	30	31

left when added to the partial product. This right 2 shift is accomplished by shifting the partial product right 4 as it enters PAL, and shifting it left 2 when it is gated to PAB for addition to the next multiplicand multiple. The two low-order bits [B(66,67)] are shifted out, and therefore lost on each cycle.

Note: Steps 2 and 3 are performed by the 'SEL-MPL*E3' micro-order and hardware conditions.

4. Reduce E(12–15) by 1.
5. Determine, by the count in E(12–15), whether all bits of the multiplier fraction have been used to select a multiplicand multiple. If not, repeat steps 2–4; if they have all been used, proceed to step 6.
6. After decoding the last S bits, the 'TX' trigger is checked. If the 'TX' trigger is set, one additional termination cycle is necessary to obtain the final intermediate product. If the 'TX' trigger is reset, no extra cycle is necessary. After the fraction intermediate product is obtained, the fraction is normalized (post-normalization), the characteristic is adjusted, the sign is determined, and the final 64-bit product is stored into the FPR specified by R1 and R1 + 1 [located in E(8–11)].

To illustrate the multiply operation, assume that the following fractions are to be multiplied:

$0.24_{10} \times 0.15_{10} = 0.0360_{10}$ or $0.18_{16} \times .F_{16} = 0.168_{16}$

The operands in machine language become:

0 1000000. 0001 1000 0↔0 X 0 1000000. 1111 0000 0↔0

In hex notation, the example becomes:

+40.18 X +40.F = +40.168

Further, assume that a short operand instruction in the RR format is to be executed. For this discussion, assume that 0.15 (decimal) is the multiplier (first operand) and 0.24 (decimal) is the multiplicand (second operand). At the start of execution, the instruction is in E; the first operand is in A, B, and D (this value is not used and is subsequently destroyed), and the second operand is in S and T.

The signs are saved, the characteristics are determined, and the fractions are prenormalized before beginning the multiply algorithm. The value in E(12–15) is set to 15 and sequentially reduced by 1 during the operation. Before fraction multiplication begins, the first operand fraction is transferred to S, the second operand is transferred to D, and B and T are reset.

The first multiple of the multiplicand is determined by checking E(12–15), which initially contains 1111 (binary). Using Table 3-10 to determine the M1 and M2 bits, the first bits selected are S(30,31). At this time, S(30,31) = 00. Thus, the first partial product placed into AB equals zero. The sequential reduction of E(12–15) continues until the value equals 0101, at which time the partial product in AB equals zero.

Referring to Table 3-10, when E(12–15) = 0101, S(10,11) is selected. These selected bits determine the multiple (M1, M2) of the multiplicand to be added to the partial product in AB. Because S(10,11) = 11 (binary), the 2's complement of DT is gated to PAA. The contents of AB are shifted left 2 at this time (AB equals zero) and gated to PAB. The output of the parallel adder is shifted right 4 to the PAL's. The contents of the PAL's are gated to AB, forming a new partial product. AB(4–67) now contains 1111.1111 1110 1000 0↔0. PA(4) is propagated into PAL(4–7) by the 'R→' micro-order. Because S(10,11) = 11, the 'TX' trigger is set (Table 3-11). Because E(12-15) is decremented after each multiple selection, zeros are added to PAA on the next multiple selection (0 X DT), as shown in Table 3-11. During this select multiple, the contents of AB are shifted left 2 to PAB, and the next partial product is shifted right 4 to the PAL's and AB(4–67), thus yielding an effective right 2 shift. The new partial product in AB(4–67) becomes 1111.1111 1111 1010 0↔0.

Table 3-11. Value of Multiple Determined by Multiple Selection Bits (Floating-Point)

Multiple Selection Bits		TX	DT Register Times Value Indicated (Add	Set TX
M1	M2	Trigger	to Partial Product in AB)	Trigger
0	0	0	0 X DT	No
0	1	0	1 X DT	No
1	1	0	2 X DT	No
1	1	0	-1 x DT (2's Complement)	Yes
0	0	1	1 X DT	No*
0	1	1	2 X DT	No
1	0	1	-1 x DT (2's Complement)	Yes
1	1	1	0 X DT	Yes

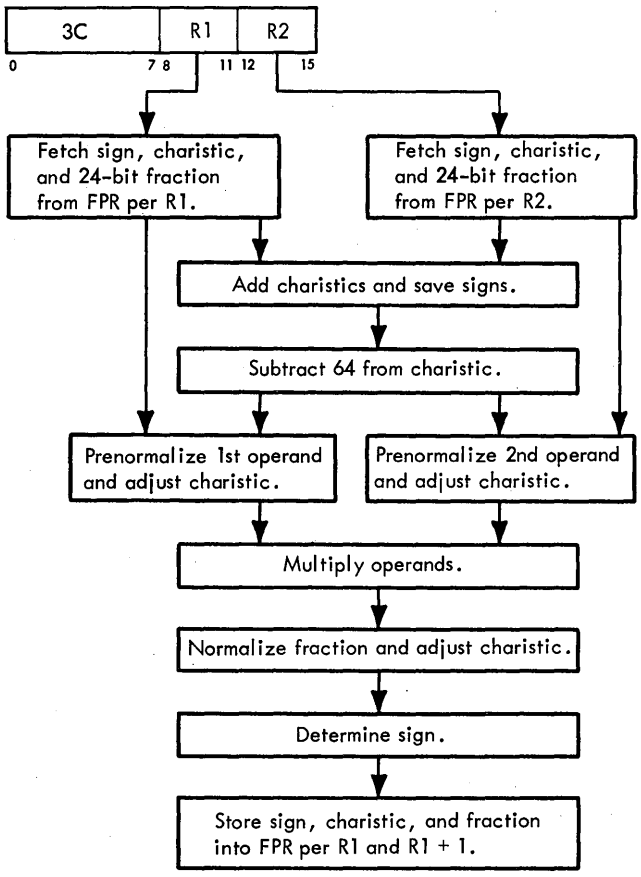
*Used on last multiple selection if 'TX' trigger is set.

At this point, all multiples of the multiplicand have been selected. If the 'TX' trigger is not set, the partial product in AB becomes the intermediate result. In this example, however, the 'TX' trigger was set because the multiplier bits equalled 11 and the 'TX' trigger was previously set (Table 3-11). Therefore, DT must be added to the partial product in AB. The contents of AB are shifted left 2 to PAB and added to DT. No right 4 shift from the parallel adder to the PAL's occurs at this time. The intermediate product is transferred from the PAL's to AB(4–67) and DT. The intermediate product is .0001 0110 1000 0↔0 (0.168_{16}). In this example, normalization is not necessary. The sign, characteristic, and fraction are stored into the FPR specified by R1 and R1 + 1. An end-op cycle completes the operation.

If the integers were preceded by zeros in this example, prenormalization of the operands would occur before executing the multiple algorithm.

Multiply, MER (3C) – RR Short Operands

- Multiply 1st operand (in FPR, per R1) and 2nd operand (in FPR, per R2) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Multiply, MER, instruction (Diagram 5-211, FEMDM) multiplies the second operand (specified by R1) by the first operand (specified by R2) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

The second operand fraction (multiplicand) is transferred from T to D. The first operand sign is saved in STAT F and the second is saved in STAT C. The characteristics are added, yielding an excess-128 characteristic, and the sum is placed into F. SA(0) is saved in STAT D and placed into F(0). B and T are reset by transferring zeros from PAL(32–63) to B and T. The first operand (multiplier) is fetched from LS and placed into S to be used for the select multiple function. A constant of 15 is placed into E(12–15) to be used for selecting the two multiple bits from S. The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested, via a four-way branch, to determine whether prenormalization is necessary. The four-way branch tests for the following conditions:

1. First and second operands are normalized.
2. First operand is normalized and the second is unnormalized.
3. First operand is unnormalized and the second is normalized.
4. First and second operands are unnormalized.

Assume that both operands need normalizing. The second operand is normalized by left-shifting the fraction in DT one hex digit and subtracting 1 from the characteristic. Left-shifting continues until the second operand fraction is normalized.

After the fraction of the second operand is normalized, the first operand (multiplier) is transferred from S to B. The contents of T (0's for short operands) are saved in the LSWR. Normalization is accomplished by left-shifting the contents of AB one hex digit and subtracting 1 from the characteristic. (B is reset during the first shift.) Left-shifting continues until the fraction of the first operand is normalized. On each left-shift, the shifted low-order fraction (0's) is stored into the FPR specified by R1

[E(8–11)]. S is then loaded with the short operand multiplier. T is reset, and DT becomes a 56-bit multiplicand (second operand).

Because the characteristic is in excess-128 notation, 64 is subtracted from F so that the excess-64 rule applies. AB is reset, and the multiply algorithm begins. A 'SEL-MPL*E3' micro-order is executed, and 1 is subtracted from E(12–15) for each machine cycle. When E(12–15) = 0100, all 12 pairs of multiples have been selected. Because the 'TX' trigger may have been set on the previous multiple selection, a last multiple selection is necessary to add in the multiplicand to obtain the correct product.

Because the operands were normalized before multiplying, a maximum of one left-shift is necessary to normalize the intermediate product fraction. If A(8–11) = 0, one left shift of the intermediate product fraction is necessary. When the left shift occurs, a 1 is subtracted from the characteristic sum. The characteristic of the final product is located in SAL(1–7) and F(1–7). The sign is determined algebraically, and then the sign, characteristic, and 56-bit fraction are stored into the FPR specified by R1 and R1 + 1.

If SAL(0) = 1, an exponent overflow or exponent underflow condition exists and the product is incorrect. Zeros are stored into the first operand location if an exponent underflow has occurred. A program interruption occurs on all exponent overflows, and on exponent underflows if masked on. If SAL(0) = 0, the stored product is correct. An end-op cycle completes instruction execution.

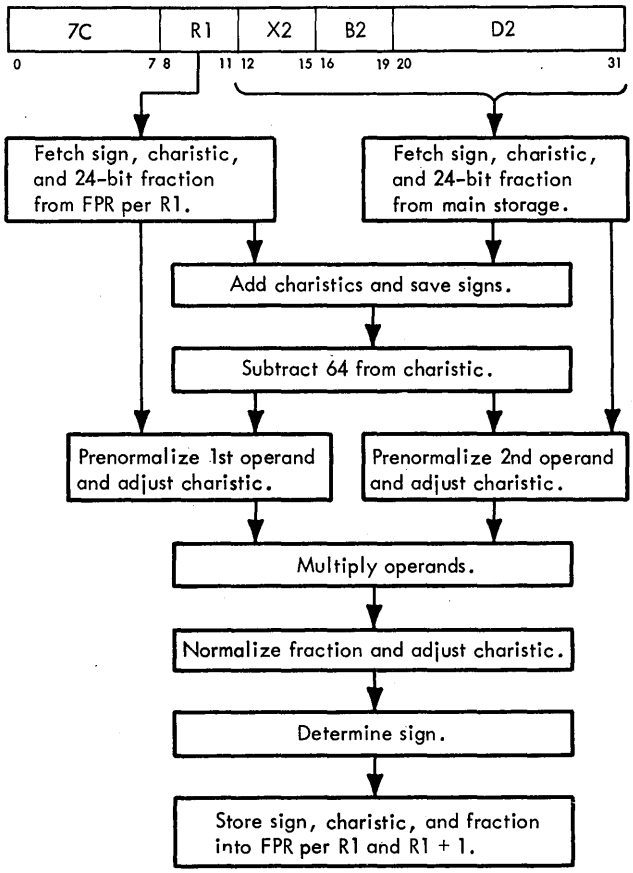
Multiply, ME (7C) – RX Short Operands

- Multiply 1st operand (in FPR, per R1) and 2nd operand (in storage) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RX format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Multiply, ME, instruction (Diagram 5-211) multiplies the second operand (specified by the effective address) by the first operand (specified by R1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The first 16 bits of the instruction are in E.



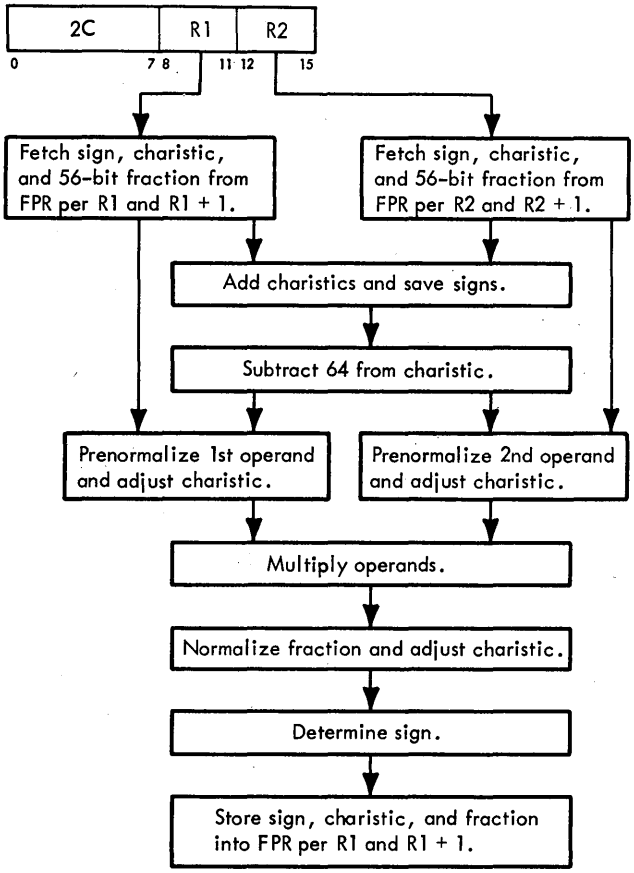
The first operand (multiplier) is placed into A, a constant of 15 is placed into E(12–15), and the STC is set to 4. Assume that the first operand is not normalized and that the second is normalized. The second operand is fetched from main storage (per the effective address in D) and placed into ST. If D(21) = 1, the second operand (multiplicand) is in T; conversely, if D(21) = 0, the second operand is in S and must be placed into T. The sign of the first operand is saved in STAT F and the sign of the second is saved in STAT C. The characteristics are added, and the sum is placed into F. SAL(0) is saved in STAT D and F(0). The fraction of the second operand is placed into D. B and T are reset, the first operand is placed into S, and 15 is placed into E(12–15). A four-way branch determines the next operation. From this point, operation is similar to that of the MER instruction.

If the first operand was normalized, the second operand (multiplicand) from main storage is placed into AB. T and the STC are reset. The transfer of the second operand fraction to D is determined by D(21). If D(21) = 1, the second operand from B is transferred to A and D. If D(21) = 0, the second operand in A is transferred to D. Note that the sign of the first operand is saved in STAT C and that of the second in STAT F. The characteristics are added, and the sum is saved in F. The characteristic carry is saved in STAT D and F(0).

Because the first operand was initially normalized, the ROS microprogram assumes that the second operand is also normalized. Therefore, the first partial product is computed. If the second operand needs normalizing, however, the operands and the constant 15 in E(12–15) are restored, and the second operand is normalized before multiplying. Once both operands are normalized, the operands are multiplied and the results stored.

Multiply, MDR (2C) – RR Long Operands

- Multiply 1st operand (in FPR, per R1 and R1 + 1) and 2nd operand (in FPR, per R2 and R2 + 1) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Multiply, MDR, instruction (Diagram 5-212, FEMDM) multiplies the second operand (specified by R2 and R2 + 1) by the first operand (specified by R1 and R1 + 1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

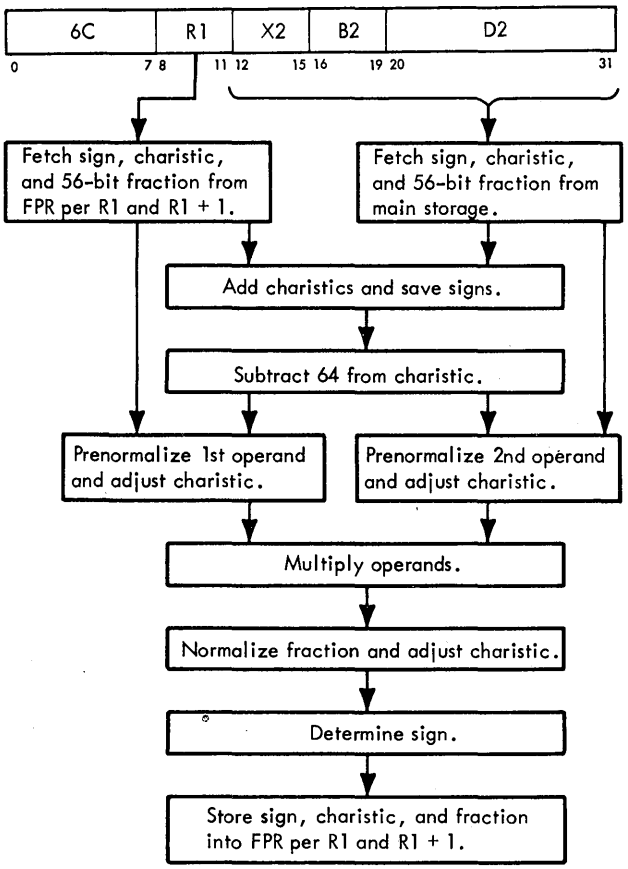
The high-order fraction of the second operand (multiplicand) is transferred from T to D. The low-order fraction of the first operand (multiplier) is placed into S, and that of the second operand is placed into T. DT contains the multiplicand fraction, and S contains the low-order fraction of the multiplier. The signs are saved in STAT C and STAT F. The characteristics are added, and the sum is placed into F. The characteristic carry is saved in STAT D and also placed into F(0). A constant of 15 is placed into E(12–15) to be used for selecting the two multiple bits located in S.

The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested, via a four-way branch, to determine whether prenormalization is necessary. Assume that the first operand is normalized and that the second needs normalizing. The second operand is normalized by left-shifting the contents of DT one hex digit and subtracting 1 from the characteristic on each shift. Left-shifting continues until the fraction is normalized. Because the characteristic sum is in excess-128 notation, 64 is subtracted from the characteristic in F. AB is reset, and the first multiple is selected. The multiples are selected per E(12–15) until E(12–15) = 0001, indicating that the multiples must be selected from the high-order fraction located in LS. Accordingly, the high-order fraction of the first operand (multiplier) is fetched from LS per R1 [E(8–11)], and placed into S. From this point, the multiply execution is the same as that for the MER instruction.

Multiply, MD (6C) – RX Long Operands

- Multiply 1st operand (in FPR, per R1 and R1 + 1) and 2nd operand (in storage) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Multiply, MD, instruction (Diagram 5-212) multiplies the second operand (in storage) by the first operand (specified by R1 and R1 + 1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The low-order fraction of the first operand is in LS.
3. A main storage request for the second operand has been issued per the effective address in D.
4. The first 16 bits of the instruction are in E.

The first operand (multiplier) sign, characteristic, and high-order fraction are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed into S. If the first operand is not normalized, the STC is reset, the low-order fraction is transferred from S to B, the second operand (multiplicand) is fetched from main storage and placed into ST and D (high-order fraction in D), the characteristics are added, and the signs are saved (first operand sign in STAT F and second in STAT C).

The low-order fraction of the first operand is again placed into S, and a constant of 15 is placed into E(12--15). The four-way branch determines the next operation. The remainder of the operation is identical to that of the MDR instruction.

If the first operand was normalized, the second operand is fetched from main storage and placed into AB. The second operand fraction (multiplicand) is transferred from AB to DT. The sign of the first operand is saved in STAT C and that of the second in STAT F. The characteristics of the first and second operands are added; the result is placed into F, and the characteristic carry is saved in STAT D. [The carry is also transferred to F(0).] Because the first operand is normalized, the ROS microprogram assumes that the second operand is also normalized; therefore, the first multiple is selected. If the second operand needs to be normalized, the initial conditions are restored and the ROS microprogram proceeds with the normalization of the second operand.

DIVIDE

- Divides 1st operand (dividend) by 2nd operand (divisor) and places normalized quotient into 1st operand location.
- Characteristics are subtracted, and 64 is added to the characteristic difference.
- Operands are prenormalized before dividing.
- Quotient is 32 bits for short operands, 64 bits for long operands.
- Quotient is normalized.
- Sign of quotient is determined algebraically.
- No remainder is retained.

The Divide instruction divides the first operand (dividend) by the second operand (divisor) and places the normalized quotient into the first operand location. In short operand instructions, the low-order halves of the FPR's are ignored and remain unchanged.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics, plus 64, is used as an intermediate quotient characteristic.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift of one hex digit may be necessary if the normalized dividend fraction is larger than the normalized divisor fraction. The intermediate quotient

characteristic is adjusted for the shifts. Low-order digits of the quotient fraction are removed to obtain the desired number of digits.

The sign of the quotient is determined algebraically. That is, if the signs of the operands are alike, a plus sign is assigned to the quotient; if the signs are unlike, a minus sign is set into the quotient.

A program interruption for exponent overflow occurs when the final quotient characteristic exceeds 127; the operation is terminated.

A program interruption for exponent underflow occurs when the final quotient characteristic is less than zero and the corresponding mask bit is a 1. Underflow is not signalled for the intermediate quotient or for the operand characteristics during prenormalization.

If division by a divisor with a zero fraction is attempted, the divide operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient fraction will be zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interruption for exponent underflow or exponent overflow. The program interruption for significance is never taken for division. The CC remains unchanged.

Characteristic Computation

After the first and second operands are fetched and placed into the proper registers, the characteristics are subtracted. Because the complement gates to the serial adder are on the SAA bus, the first operand characteristic (C1) is subtracted from the second operand characteristic (C2). Therefore, the characteristic computations differ from what might be expected. (Normally, C1 - C2 would be expected.)

The CE takes the following steps in computing the quotient characteristic:

1. Subtracts C1 from C2 (dividend characteristic from divisor characteristic).
2. Subtracts 64 from the characteristic difference.
3. Normalizes the first operand and adds a 1 to the intermediate characteristic for each digit position that the fraction is shifted.
4. Normalizes the second operand and subtracts a 1 from the intermediate characteristic for each digit position that the fraction is shifted.
5. Takes the 2's complement of the intermediate characteristic.
6. Checks for a divisor fraction greater than a dividend fraction. If the dividend is the larger number, right-shifts the dividend one hex digit and adds 1 to the characteristic.
7. Saves the final characteristic.
8. Checks the final characteristic for exponent overflow or exponent underflow.

As an example of this computation, assume that two hex numbers are to be divided, .004 by .02:

$$\frac{\text{1st operand dividend}}{\text{2nd operand divisor}} \pm \frac{.004 \times 16^5}{.02 \times 16^2} = \pm .2 \times 16^3 = \pm 3.2$$

Fraction

Characteristic

Convert the above characteristics to excess 64 notation:

C1

$$\frac{69.004}{66.02}$$

C2

Convert the above characteristics to binary form:

$$\frac{1000101.004}{1000010.02} = \frac{69.004}{66.02} = \pm 67.2 \text{ or } .2 \times 16^3$$

after 64 is subtracted from the characteristic.

Step 1. The machine subtracts the characteristics (C2 - C1):

```
1000010      C2
0111010  }   2's complement of C2
      1  }
1111101
```

$$\frac{.004}{1111101.02}$$

Step 2. 64 is subtracted from the characteristic to maintain excess-64 notation:

```
1111101
0111111  }   2's complement of 64
      1  }
0111101
```

$$\frac{.004}{0111101.02}$$

Step 3. Note that the first operand hex fraction requires two left-shifts to prenormalize. Shift left 2 and add 2 to the characteristic:

```
0111101
0000010
0111111
```

$$\frac{.4}{0111111.02}$$

Step 4. The second operand hex fraction requires one left-shift. Shift left 1 and subtract 1 from the characteristic:

```
0111111
1111110  }   2's complement of 1
      1  }
0111110
```

$$\frac{.4}{0111110.2}$$

Step 5. Take the 2's complement of the characteristic:

```
1000001  }   2's complement of 0111110
      1  }
1000010
```

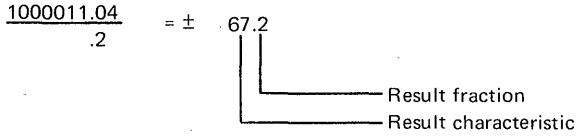
$$\frac{1000010.4}{.2}$$

Step 6. Because the dividend fraction is greater than the divisor fraction, the dividend is shifted right 1 and 1 is added to the characteristic before dividing fractions:

```
1000010
0000001                                    1000011.04
1000011      = 67 = final characteristic       .2
```

Step 7. Save 67, which is the final characteristic.

Step 8. Divide fractions and store quotient:



Subtracting the first operand characteristic from the second effectively makes the characteristic difference part of the divisor (dividend/divisor); to add to the characteristic, therefore, the value must be subtracted. For example, excess-64 notion is used in the CE. Subtracting (C1 + 64 from C2 + 64) equals C2 - C1 + 0; therefore, 64 must be added to the characteristic difference to maintain excess-64 notation. Because the C2 minus C1 difference is 2's-complemented later in the operation, 64 must be subtracted (2's complement and add) from the characteristic that is part of the divisor. The characteristic must be part of the dividend to obtain the final quotient characteristic.

The intermediate characteristic is 2's-complemented to obtain the correct characteristic of the quotient because the initial characteristic subtraction places the intermediate characteristic in the divisor. The intermediate characteristic is not considered to be in the 2's complement form.

Normalization

In the divide operation, both fractions must be normalized before dividing the fractions. Also, the divisor must be larger than the dividend. If the divisor is less than the dividend, the dividend is divided by 16 by right-shifting the dividend four binary bit positions. Prenormalization and making the divisor larger than the dividend make postnormalization unnecessary.

Fraction Division

- Is performed as follows:
  1. Gate dividend to adder, shifted left 2; gate divisor to adder, shifted left 1.
  2. Add; result is partial dividend.
  3. Develop quotient bit: if partial dividend is in true form, place a 1 into quotient; if partial dividend is in 2's complement form, place a 0 into quotient.
  4. Gate partial dividend and divisor to adder (no shift). If partial dividend is in true form, gate divisor in 2's complement form; if partial dividend is in 2's complement form, gate divisor in true form.
  5. Perform steps 2 and 3.
  6. Perform steps 1–5.
  7. Continue until count signals end of algorithm.

The basic algorithm for the floating-point fraction divide operation is similar to the algorithm used in fixed-point divide. The characteristics of the two operands are subtracted, and 64 is added to maintain excess-64 notation. The divisor fraction is subtracted from the dividend fraction. A carry indicates that the dividend is greater than the divisor. The dividend must be less than the divisor; if not, a right 4 shift of the dividend is required. Division is accomplished by successive subtractions and storing of quotient bits as determined by the carry. Successive subtractions are performed, and the divisor is, in effect, shifted right one position with respect to the dividend for each subtraction (actually, the dividend is shifted left as explained below).

In binary arithmetic, to divide one number (dividend) by another (divisor), the dividend is repeatedly reduced by subtracting the divisor. The number of times this reduction can be done is the solution (quotient). There are two methods of performing binary division: restore and nonrestore (Figure 3-11).

In restore division, the result of a reduction of the dividend by the divisor is retained only if the result is the true difference (as opposed to a 2's complement difference); a carry indicates that the result is in true form. This result, called the partial dividend, is used in the next reduction. However, if the result is the 2's complement of the difference (no carry), the result is discarded and the old partial dividend is doubled in relation to the divisor to participate in the next reduction. A 1 is inserted into the quotient when the result is true (carry), and a 0 is inserted when the result is in 2's complement form (no carry).

In the nonrestoring method of division, the result of a reduction is retained as the new partial dividend whether it is in true or 2's complement form. When a partial dividend is in true form, the 2's complement of the divisor is added to it; when the partial dividend is in 2's complement form, the true divisor is added to it. In each reduction, the partial dividend is shifted left one bit in relation to the divisor; also, a 1 is inserted into the quotient when the result is true (carry) and a 0 is inserted into the quotient when the result is in 2's complement form (no carry).

Shifting the dividend left one position doubles its value and is equivalent to halving the divisor. Similarly, shifting the dividend left two positions quadruples it and is equivalent to reducing the divisor to ¼ of its value. Note

Problem: 45 ÷ 7 = 6 3/7

Dividend: 0 0 1 0 1 1 0 1 = (2 x 16) + (13 x 1) = 45
Divisor: 0 1 1 1 = (7 x 1) = 7
Quotient: 0 1 1 0 = (6 x 1) = 6
Remainder: 0 0 1 1 = (3 x 1) = 3

n/c = no carry
c = carry

A. Restore

```
                    00110
              0111 | 00101101
2's Complement Divisor:        1001          1st Reduction
2's Complement Result (Discarded):  n/c 1011

True Partial Dividend:          0101
2's Complement Divisor:         1001          2nd Reduction
2's Complement Result (Discarded):  n/c 1110

True Partial Dividend:          1011
2's Complement Divisor:         1001          3rd Reduction
True Result:                  c 0100

True Partial Dividend:          1000
2's Complement Divisor:         1001          4th Reduction
True Result:                  c 0001

True Partial Dividend (Remainder):  0011
2's Complement Divisor:             1001      5th Reduction
2's Complement Result (Discarded):  n/c 1100
```

B. Non-Restore

```
                    00110
              0111 | 00101101
2's Complement Divisor:        1001          1st Reduction
2's Complement Result:      n/c 1011

2's Complement Partial Dividend:  0111
True Divisor:                     0111        2nd Reduction
2's Complement Result:         n/c 1110

2's Complement Partial Dividend:    1101
True Divisor:                       0111      3rd Reduction
True Result:                     c 0100

True Partial Dividend:              1000
2's Complement Divisor:             1001      4th Reduction
True Result:                     c 0001

True Partial Dividend (Remainder):  0011
2's Complement Divisor:             1001      5th Reduction
2's Complement Result:           n/c 1100

2's Complement Partial Dividend (Remainder-Divisor):  1100
True Divisor:                                         0111   Correction Cycle
True Result (Remainder):                           c 0011
```

Figure 3-11. Restore and Non-Restore Division

the similarity between the restore and nonrestore methods of division (the first successful reduction of the dividend is made by one-quarter of the divisor):

(Restore) Dividend minus ¼ Divisor, is equivalent to (Non-Restore) Dividend minus Divisor + ½ Divisor + ¼ Divisor.

In the 7201, the nonrestoring method of division is used, and the dividend is shifted left rather than shifting the divisor right. The development of the quotient and the left-shifting of the dividend is performed as follows:

1. The dividend is gated to the adder shifted left 2. The divisor is gated to the adder shifted left 1, thus yielding an equivalent right 1 displacement with respect to the dividend. (The divisor may be in true or 2's complement form, depending on the partial dividend.)
2. The two numbers are added, and the result (partial dividend) is placed back into the dividend register.

3. A quotient bit is developed from the partial dividend obtained in step 2. If the partial dividend is in true form, a high-order carry occurred and the high-order bit is a 0; a 1 is therefore placed into the quotient. Conversely, if the partial dividend is in 2's complement form, a high-order carry did not occur and the high-order bit is a 1; a 0 is therefore placed into the quotient.
4. The partial dividend and the divisor are gated to the adder. Because the partial dividend was shifted left 2 in step 1, the correct displacement between them exists, and they are not shifted now. If the partial dividend obtained in step 2 is in true form, the divisor is gated in 2's complement form; if the partial dividend is in 2's complement form, the divisor is gated in true form.
5. Steps 2 and 3 are repeated.
6. Steps 1 through 5 are repeated.

The operation continues developing quotient bits, developing new partial dividends, gating the partial dividend to the adder (shifted left 2 every other time), and gating the divisor to the adder in true or 2's complement form (shifted left 1 every other time) until a count, which is reduced every time the partial dividend is shifted, signals the end of the algorithm.

Figure 3-12 is an example of the divide operation as it is performed in the 7201-02 CE. Note that the first subtraction does not result in a quotient bit but is used to decide whether the divisor is to be true- or 2's complement-added on the next cycle.

Data Flow and Algorithm

- Signs are saved in STAT's C and F.
- Characteristic computation is performed in serial adder.
- Characteristic difference is saved in F.
- 'DVDL0' and 'DVDL1' micro-orders are unique to divide algorithm.

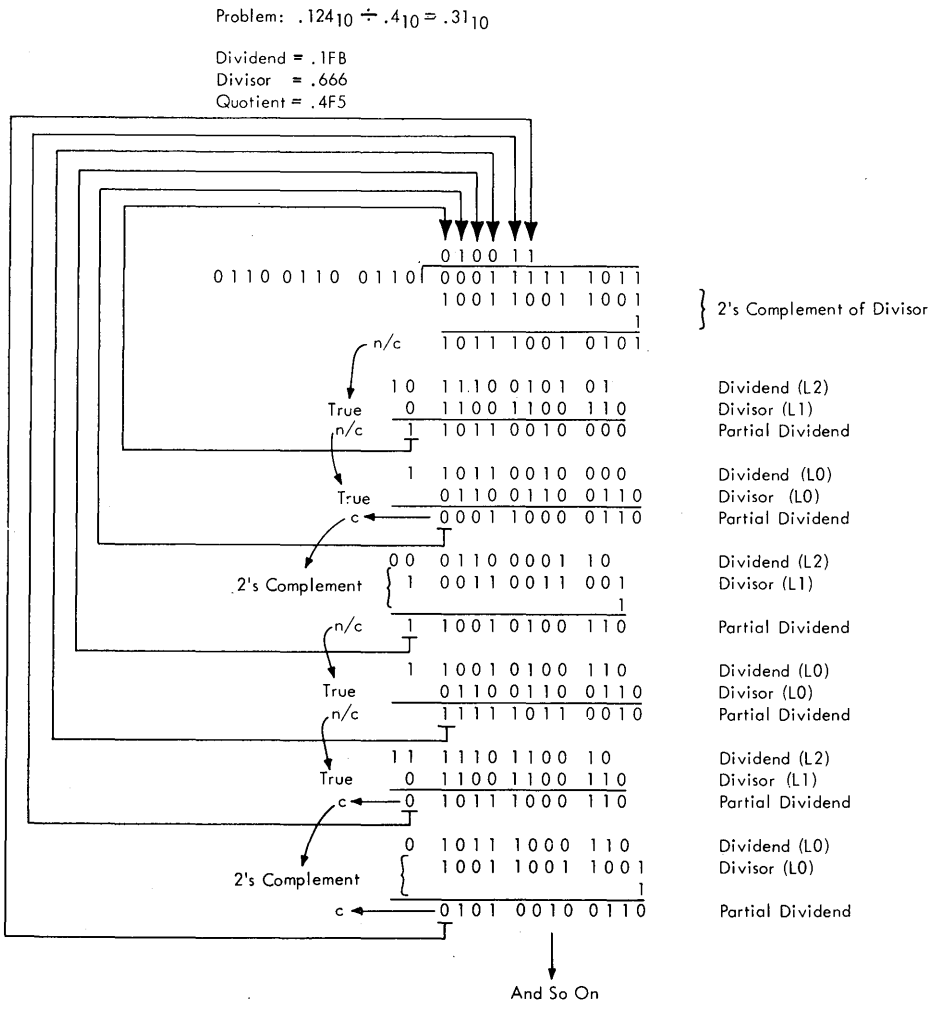


Figure 3-12. Fraction Divide Example

- Fraction division uses parallel adder.
- Divisor is in DT; dividend is in AB; quotient is developed in S.

The first operation that occurs is the computation of the final characteristic (For an example, see "Characteristic Computation"). The data paths for the signs and characteristics are shown in A of Diagram 5-213, FEMDM. The signs are saved in STAT C and STAT F. The first and second operand characteristics are gated to the SAA and SAB per the ABC and STC, respectively. To subtract the characteristics, the 2's complement of the first operand characteristic is added to the second operand characteristic. The characteristic difference is stored into F(0–7), and the characteristic carry [SA(0)] is saved in STAT D. Other inputs to the SAB bus allow subtracting 64, subtracting 1, gating the 2's complement of F, or adding 1 to the value in F. After the final characteristic is computed, the result is stored into S(0–7) per the STC.

The data path for the derivation of the divide multiple is shown in B of Diagram 5-213. When the divide algorithm begins, the divisor (first operand) is in DT and the dividend is in AB.

Two micro-orders ('DVDL0' and 'DVDL1') are used specifically during the divide algorithm. These micro-orders have three functions: (1) to gate the true or 2's complement of DT (divisor) to the PAA; (2) to determine the amount of shift (L0 = no shift, L1 = left 1 shift) of the divisor (contents of DT) to the PAA; and (3) to determine the partial quotient (PQ) bit and the PQ bit location after the subtraction of the divisor and the partial dividend has taken place.

Whether the true or 2's complement form of the divisor is sent to the PAA is determined by the PA(4) carry from the previous algebraic subtraction of the divisor and partial dividend, and the amount of shift (L0 or L1) as determined by the 'DVDL0' or the 'DVDL1' micro-order. If a PA(4) carry occurred, the 2's complement is gated (L0 or L1) to the parallel adder. If a PA(4) carry did not occur, DT is gated (L0 or L1) in true form to the parallel adder. The data in AB is gated to PAB with no shift or a left 2 shift under micro-order control ('AB' and 'ABL2')

As previously noted, the 'DVDL0' and the 'DVDL1' micro-orders determine the PQ bit and the location of the bit. The PQ bit is determined by testing AB(4) for a 0 or a 1. If AB(4) = 1, the partial dividend is in 2's complement form and a 0 is placed into the selected PQ SAL location. If AB(4) = 0, the partial dividend is in true form, and a 1 is placed into the selected PQ SAL location.

As shown in B of Diagram 5-213, the PQ location in SAL is determined by E(14,15) and by the 'DVDL0' or 'DVDL1' micro-order. E(14,15) selects the pair of SAL bits into which the PQ bit is to be placed. The 'DVDL0' micro-order selects the odd bit of the selected pair; the 'DVDL1' micro-order selects the even bit. At the same time that the PQ bit is gated into SAL, the contents of F are added to the PQ bit and saved in F. After a PQ byte (eight bits) is available, the contents of F(0–7) are gated to S per the STC. After S is filled with the quotient (or PQ), the contents of S are stored into LS per E(8–11).

For a discussion of the divide algorithm, assume that the final characteristic is in S(0–7) and that the normalized fractions are in DT (divisor) and AB (dividend). By definition, the CE requires that floating-point numbers consist of a sign, a characteristic, and a fraction. Because no provisions are made in the CE to handle integers in floating-point instructions, the divisor must be larger than the dividend to retain a fraction quotient. After both fractions are normalized, therefore, the contents of DT are subtracted from the contents of AB. A carry from PA(4) indicates that the dividend is larger than the divisor. Whenever the dividend is larger than the divisor, the contents of AB must be restored and shifted right 4 (divided by 16) before proceeding with the divide algorithm, and a 1 must be added to the characteristic. If no carry occurred from PA(4), the dividend is less than the divisor, and the CE proceeds with the divide algorithm.

When the divisor (d) is subtracted from the dividend (D), the difference is placed into AB (D - d in AB). If a right 4 shift was necessary, the divisor (d) is restored and divided by 16 (d in DT). AB now contains the dividend (D). At the beginning of the divide algorithm, the 2's complement of DT is shifted left 1 and added to the contents of AB shifted left 2; the result is placed into AB, thus yielding an effective left 1 shift of AB (dividend). The contents of AB may be expressed by the equation 4D - 2d = contents of AB. The value 4D - 2d is in AB after the first machine cycle of the divide algorithm.

If the dividend was less than the divisor, D - d is in AB. The CE proceeds to add the contents of DT shifted left 1 to the contents of AB shifted left 2, with the result placed into AB. This addition results in the equation 4(D - d) + 2d = contents of AB. Simplifying the equation yields 4D - 4d + 2d = 4D - 2d. At the end of the first cycle of the divide algorithm, the same result (4D - 2d) is obtained as when the dividend was larger than the divisor. The CE continues with the divide algorithm.

During the first machine cycle of the divide algorithm, the 'DVDL0' micro-order also selects the DT gating to the parallel adder per the PA(4) carry. The subtraction resulting from the 'DVDL0' micro-order is accomplished during the next machine cycle. The PQ bit, however, is determined by the A(4) value that was computed during the previous machine cycle. On the first cycle of the divide algorithm, the contents of AB (dividend) are shifted left 2 by a micro-order and added to the contents of DT (divisor) shifted left 1 per a micro-order. Thus the divisor is shifted right 1 with respect to the dividend, but the resulting

partial dividend is displaced left 2 in AB. On the next divide select multiple subtraction, the dividend and the divisor are subtracted without shifting, yielding the correct right 1 displacement. The following cycle causes AB and DT to shift again. Note that, as the dividend is shifted left, the low-order bit positions of AB are filled with 0's.

As previously noted, the PQ bit is gated to SAL per E(14,15) and the 'DVDL0' or 'DVDL1' micro-order. A 1 is added to the ABC after each pair of PQ bits is gated to F via SAL. When the ABC equals 3, F contains eight PQ bits (one per byte). The PQ byte is gated to S per the STC.

After each byte is gated to S, a 1 is added to the STC. When the STC equals 3, S contains the characteristic and fraction (or high-order fraction). The contents of S are stored into the LSWR.

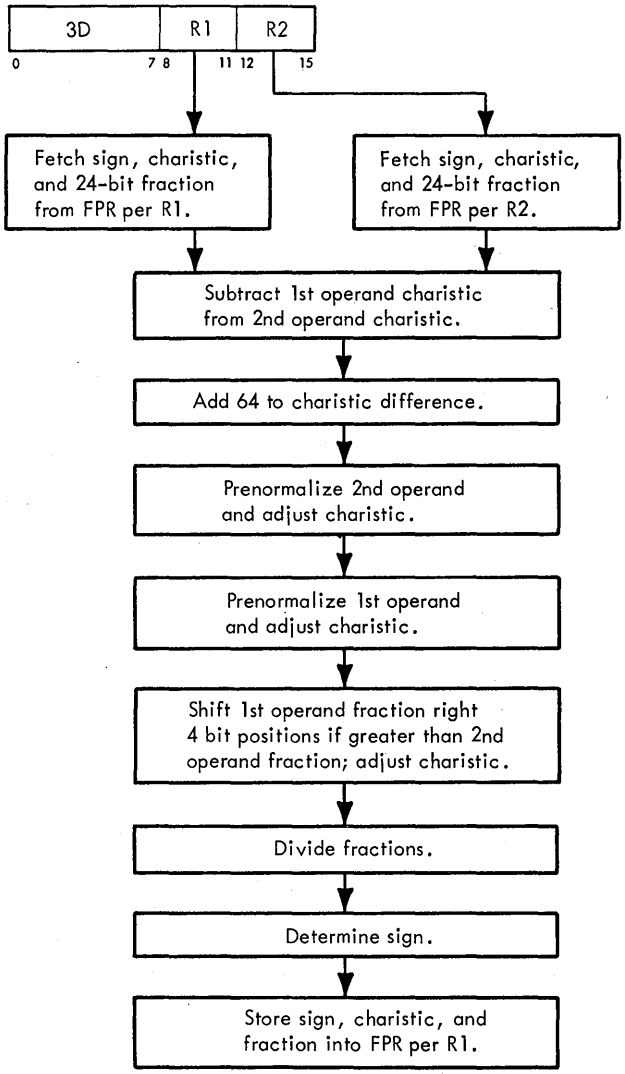
Before initiating the divide algorithm, STAT D was reset to indicate the first pass at loading PQ bytes into S. After the sign, characteristic, and high-order fraction are stored into the LSWR, the instruction and STAT D determine the next operation. If a short operand instruction is being executed, the sign is inserted and stored with the characteristic and fraction into the FPR per E(8–11). An end-op cycle completes instruction execution.

If the instruction was a long operand instruction, the sign, characteristic, and high-order fraction are stored into the FPR per E(8–11). STAT D is set, and the divide algorithm continues. The contents of the LSWR are returned to T. The same operations as described above are performed to obtain the remaining low-order fraction part of the quotient, and the same three-way branch is encountered. This time the divide algorithm is completed, and the low-order fraction is stored into the FPR per E(8–11) + 1. An end-op cycle completes instruction execution. The remainder in AB is not stored.

Divide, DER (3D) – RR Short Operands

- Divide 1st operand (in FPR, per R1) by 2nd operand (in FPR, per R2) and place normalized quotient into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Divide, DER, instruction (Diagram 5-214, FEMDM) divides the first operand (specified by R1) by the second operand (specified by R2) and places the normalized quotient into the first operand location. No remainder is retained.



The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

If no specification check occurred, the second operand fraction is transferred from T to D. The characteristics are subtracted, and 64 is algebraically added to the characteristic difference to maintain excess 64-notation. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. B and T are reset, and the contents of AB and ST are treated as 56-bit fractions.

In the divide instructions, both operands are prenormalized before the divide algorithm begins. A four-way branch determines the prenormalization path by testing A(8–11), the dividend, and PAL(40–43), the divisor, for the normalized conditions:

1. The first and second operands are normalized.

2. The first operand is normalized, and the second is unnormalized.
3. The first operand is unnormalized, and the second is normalized.
4. The first and second operands are unnormalized.

Assume that both operands are unnormalized. The second operand (divisor in DT) is shifted left 4 until the operand is normalized. A 1 is subtracted from the intermediate quotient characteristic for each shift.

After the second operand is normalized, the first operand is normalized by left-shifting until the fraction contains a hex digit [A(8–11) not equal to zero]. On each left-shift, a 1 is added to the intermediate quotient characteristic in F.

After the operands are normalized, the second operand fraction (divisor) is subtracted (take 2's complement of second operand and add) from the first operand fraction. Before branching on the PAL(4) carry, the 2's complement of the intermediate characteristic is computed and placed into F. Also, the constant 5 is placed into E(12–15) for controlling the divide algorithm. A carry from PAL(4) indicates that the dividend is larger than the divisor. If the dividend is larger than the divisor, the dividend is restored and is divided by 16 by a right-shift of one hex digit. A 1 is added to the characteristic value, which is the final characteristic of the quotient. The final characteristic is placed into S(0–7).

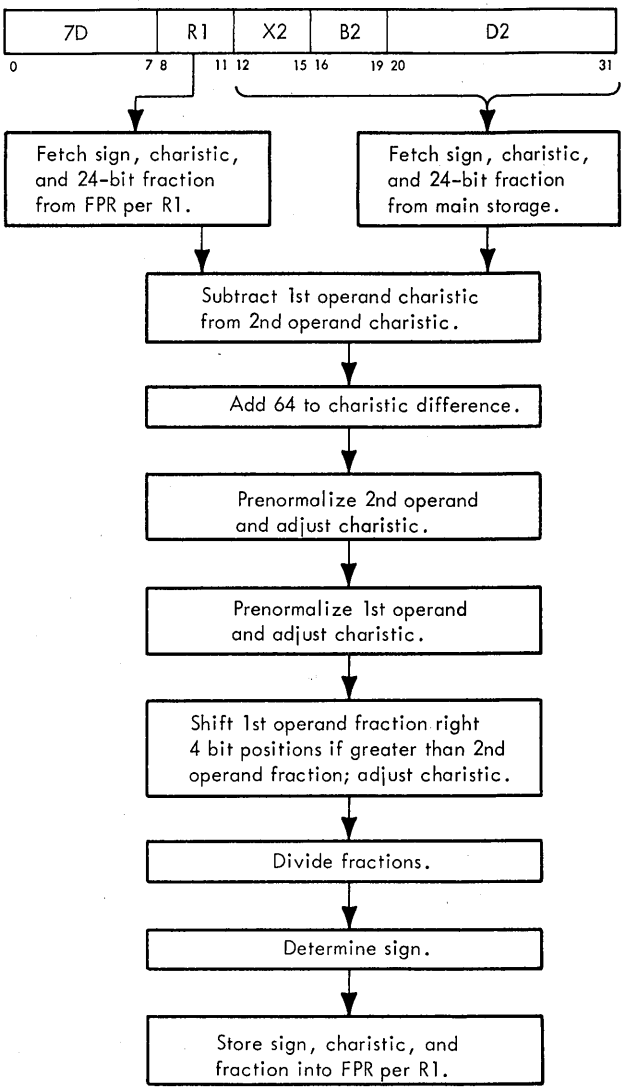
No carry from PAL(4) indicates that the dividend is less than the divisor, at which time the first machine cycle of the divide algorithm is executed. A test is made to determine an overflow or underflow condition. Assume that no overflow or underflow condition exists.

Fraction division begins as shown in Sheet 4 of Diagram 5-214. Figure 3-13 is an example of the action that occurs in parallel adder bits 4–11 (bits 12–31 being considered to equal 0's).

During the normalization routine, tests for zero fractions are made. If the second operand fraction (divisor) equals zero, the divide operation is suppressed and a floating-point divide program interruption occurs. If the first operand fraction (dividend) equals zero, a true zero quotient results. A true zero is stored into the first operand location, and an end-op cycle completes instruction execution.

Divide, DE (7D) – RX Short Operands

- Divide 1st operand (in FPR, per R1) by 2nd operand (in storage) and place normalized quotient into 1st operand location.
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Divide, DE, instruction (Diagram 5-214) divides the first operand (specified by R1) by the second operand (from main storage) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.

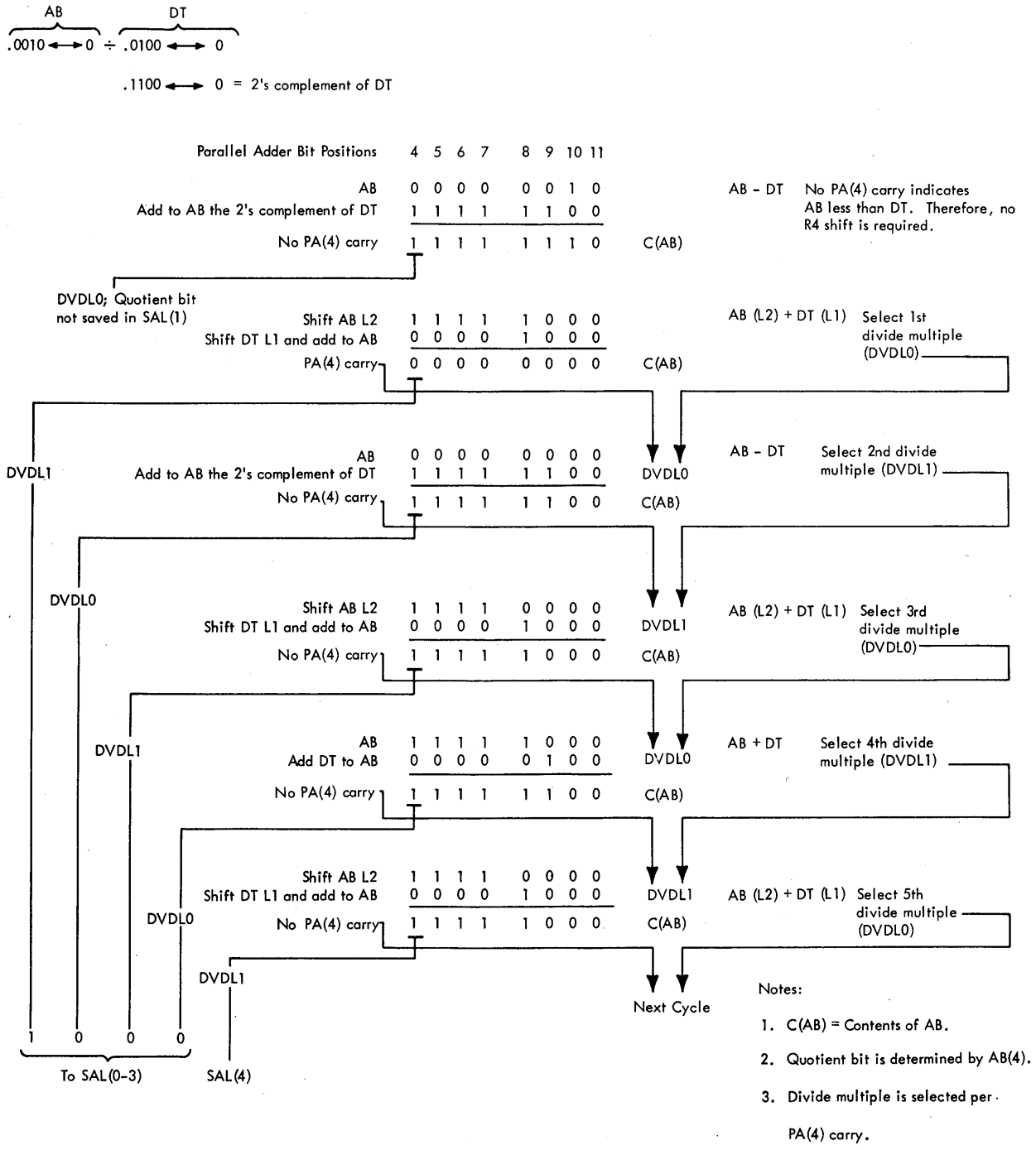


Figure 3-13. Floating-Point Divide Example

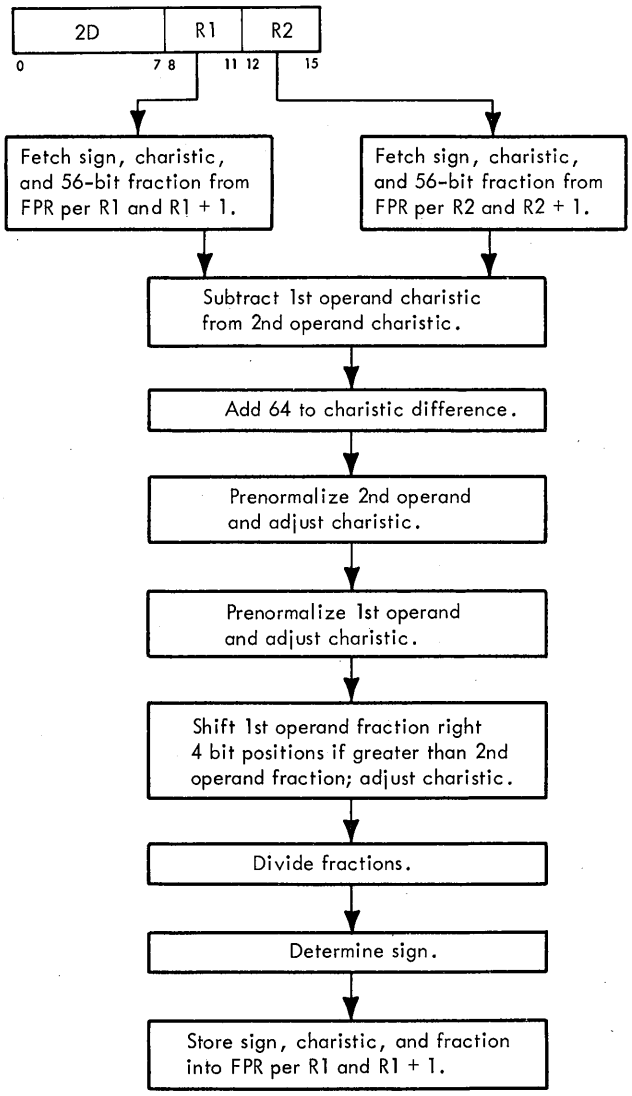
2. A main storage request for the second operand has been issued per the effective address in D.
3. The first 16 bits of the instruction are in E.

The first operand is transferred from T to A, and the STC is set to 4. The second operand is fetched from main storage per D. D(21) determines which 32 bits of the 64-bit doubleword are gated to T (Sheet 2 of Diagram 5-214). From this point, instruction execution is the same as that of the DER instruction.

Divide, DDR (2D) – RR Long Operands

- Divide 1st operand (in FPR, per R1 and R1 + 1) by 2nd operand (in FPR, per R2 and R2 + 1) and place normalized quotient into 1st operand location.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Divide, DDR, instruction (Diagram 5-215, FEMDM) divides the first operand (specified by R1 and R1 + 1) by the second operand (specified by R2 and R2 + 1) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

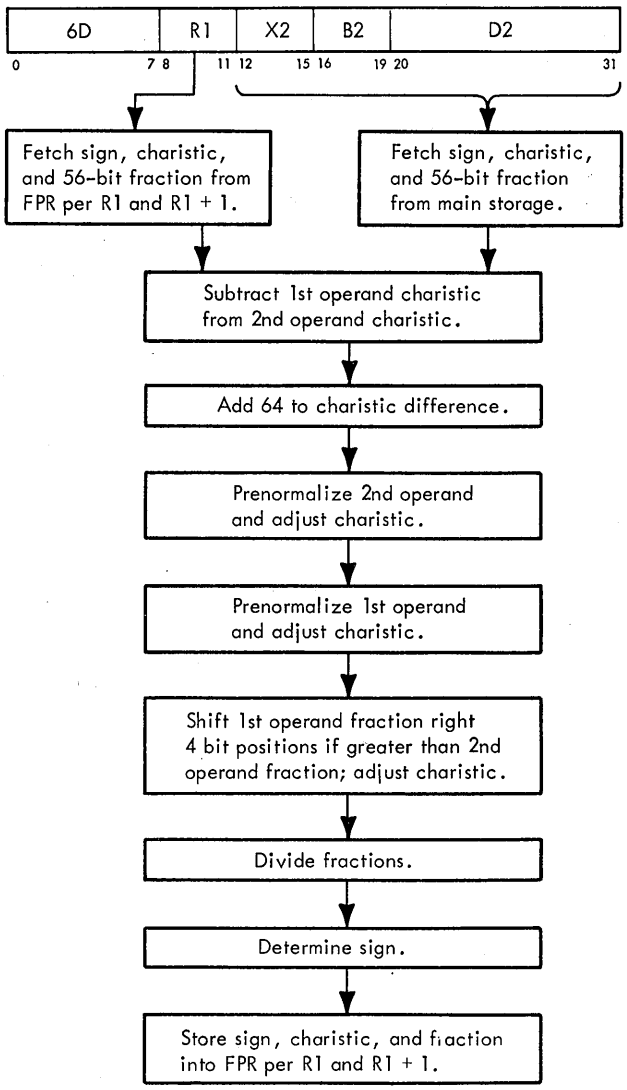
If no specification check occurred, the high-order fraction of the divisor is gated from T to D. The low-order fractions of the first and second operands are placed into B and T, respectively. As a result, the dividend fraction is in AB and the divisor fraction is in DT.

The signs are saved in STAT C and STAT F. The characteristics are subtracted, and excess-64 notation is maintained. The ABC is reset.

The four-way branch (Sheet 3 of Diagram 5-215) determines the next operation. The normalization routine, divide algorithm, and end ops are explained in the DER instruction discussion.

Divide, DD (6D) – RX Long Operands

- Divide 1st operand (in FPR, per R1 and R1 + 1) by 2nd operand (in storage) and place normalized quotient into 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Divide, DD, instruction (Diagram 5-215) divides the first operand (specified by R1 and R1 + 1) by the second operand (from main storage) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The low-order fraction of the first operand is in LS.
3. A main storage request for the second operand has been issued per the effective address in D.
4. The first 16 bits of the instruction are in E.

The sign, characteristic, and high-order fraction of the first operand are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed into B via T and the parallel adder. The second operand (64 bits) is fetched from main storage and placed into ST. The high-order fraction is transferred from S to D, and the contents of DT become the 56-bit fraction divisor. The signs are saved in STAT C and STAT F. The characteristics are subtracted, and 64 is added to the characteristic difference to maintain excess-64 notation.

The first operand (dividend) is in AB, and the second operand (divisor) is in DT. The next step is to check for the prenormalization of the dividend and divisor fractions (Sheet 3 of Diagram 5-215). The remainder of the operation is identical to that of the DDR instruction.

STORE

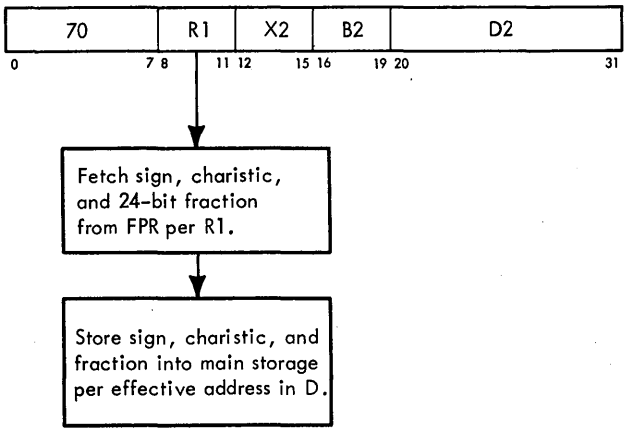
The Store instructions (STE and STD) store the first operand from an FPR in LS into the second operand location in main storage. The Store instructions are in the RX format with short and long operand options available. In the STE instruction, the low-order half of the first operand register is ignored. The first operand location remains unchanged.

Storing must be on word boundaries for the STE instruction and on doubleword boundaries for the STD instruction.

For all Store instructions, an address store compare test is made because the instructions that are in Q may be modified in main storage by the Store instruction. If an instruction is modified in main storage and is not corrected in Q, the program may not be properly executed; therefore, Q must be reloaded. The address store compare test compares the main storage address, where data is to be stored, with the effective address indicated by the Store instruction. The comparison is made by subtracting the contents of D (effective address) from the contents of the IC, shifting the difference right 4, and testing for a zero result. If the difference equals zero, the difference is less than 16; therefore, the 'program store compare' trigger is set to indicate that the instructions in Q must be refetched. The address store compare test is discussed in Section 1 of this Chapter.

Store, STE (70) – RX Short Operands

- Store 1st operand (in FPR, per R1) into 2nd operand location (in storage).
- RX format:

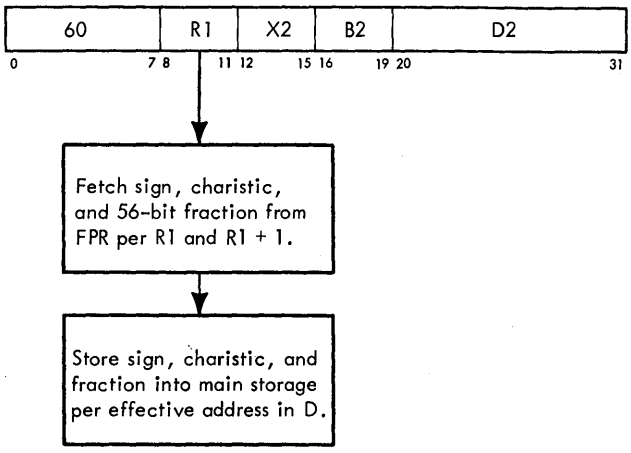


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request has been issued per effective address in D.
  First 16 bits of instruction are in E.

The Store, STE, instruction (Diagram 5-216, FEMDM) stores the first operand from the FPR specified by R1 into the main storage location specified by the effective address (second operand location). During the RX I-Fetch, the effective address is computed and placed into D, and a main storage request is initiated. The effective address must be on a word boundary or a specification program interruption is taken. Bit 21 of the effective address [PAL(61)] is tested to determine which mark triggers are set. If PAL(61) = 1, mark triggers 4–7 are set; if PAL(61) = 0, Mark triggers 0–3 are set. An address store compare test is then performed, and the instruction is terminated by an end-op cycle.

Store, STD (60) – RX Long Operands

- Store 1st operand (in FPR, per R1 and R1 + 1) into 2nd operand location (in storage).
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request has been issued per effective address in D.
  First 16 bits of instruction are in E.

The Store, STD, instruction (Diagram 5-216) stores the first operand from the even/odd pair of FPR's specified by R1 and R1 + 1 into the main storage location specified by the effective address (second operand location). During the RX I-Fetch, the effective address is computed and placed into D and a main storage request is initiated. The effective address must be on a doubleword boundary or a specification program interruption is taken. The low-order half of the first operand is gated to T from the FPR specified by R1 + 1, and mark triggers 0–7 are set. After an address store compare test is performed, the 64-bit operand is stored per the effective address, and the instruction is terminated by an end-op cycle.

SECTION 4. DECIMAL INSTRUCTIONS

This section discusses the nine instructions that operate on decimal data. The instructions use the SS format and assume packed operands and results except for Pack, which has a zoned operand, and Unpack, which has a zoned result. For a discussion of the data formats, excess-6 arithmetic, operand addressing, instruction formats, data flow, program interruptions, and condition codes, refer to Chapter 1.

INSTRUCTION HANDLING

- Depending on instruction, processing of 1 or 2 operands may be specified.
- Pack, Unpack, Move with Offset, and Zero and Add instructions operate on 1 operand.
- Add, Subtract, Compare, Multiply, and Divide instructions operate on 2 operands.
- All add-type instructions set CC.
- Major serial adder functions used by decimal instructions are:
  Excess-6 translation.
  Decimal correction.
  Complement gating.
  Cross-gating.
  Zone or sign insertion.
  Invalid digit and sign detection.
  Zero detection.

Decimal instructions may be classified into the general categories of 1- and 2-operand instructions. The 1-operand instructions are Pack, Unpack, Move with Offset, and Zero and Add. The 2-operand instructions are Add, Subtract, Compare, Multiply, and Divide.

In the 1-operand instructions, the first operand is not processed but its address is used as the destination address; the second operand is processed, and the results are placed into the first operand location. The 1-operand instructions are handled by fetching the second operand to AB. Successive AB bytes are selected per the ABC and are processed in the serial adder, and the resultant bytes are entered into ST per the STC. After all second operand bytes have been processed, the contents of ST are stored into main storage at the first operand address.

The 2-operand Add, Subtract, and Compare instructions are executed by fetching the first operand to ST and the second operand to AB. A true add or complement add operation is then performed in the serial adder one byte at a time, with the resultant bytes replacing the first operand bytes in ST as they are processed. For the Add and Subtract instructions, the results are stored into main storage at the first operand address. The Compare instruction does not store the result, but performs a test to determine the high, low, or equal relationship of the first operand to the second operand and sets the CC accordingly.

For the 2-operand Multiply and Divide instructions, the operands must be properly aligned in the registers prior to entering execution. This function is performed by the appropriate right- and left-adjust sequences incorporated in the individual microprogram of the instruction.

Basically, the multiply operation is performed by repetitive addition. The product bytes are developed one byte at a time, starting with the low-order byte. Each time one byte of product is developed, it is stored into main storage under control of the corresponding mark trigger. The instruction then proceeds to develop the next higher-order product byte. Upon execution of the instruction, the first operand is completely replaced by the product.

The Divide instruction is performed by repetitive subtraction. It is the only decimal instruction that processes the operands starting with the high-order bytes. The full divisor and a sufficient number of high-order dividend bytes are fetched to perform the first successful subtraction. Then by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient byte is developed. This byte is stored into main storage, and the instruction proceeds to develop the next lower-order quotient byte. Upon execution of the instruction, the first operand is completely replaced by the quotient and the remainder. The remainder occupies the low-order portion of the destination field.

The results of the Add, Subtract, and Compare instructions are used to set the CC. All other decimal instructions leave the code unchanged. The Add and Subtract instructions set the CC to 0, 1, or 2 to indicate a zero, less-than-zero, or greater-than-zero result; the CC is set to 3 if the result of the operation overflows. The Compare instruction sets the CC to 0, 1, or 2 to indicate that the first operand was equal to, less than, or greater than the second operand.

The serial adder performs many functions on its input data. The functions of excess-6 translation, decimal correction, and complement gating are discussed in "Data

Handling" in Chapter 1. Additional serial adder functions used by the decimal operations are:

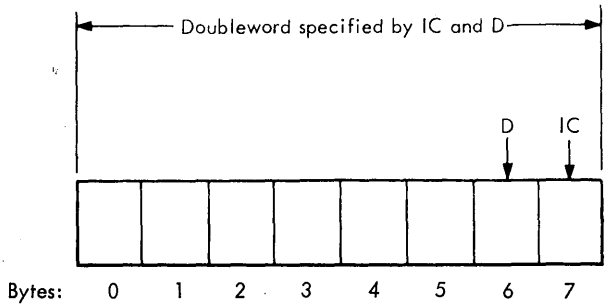
1. Cross-gating. The two-digit input to the A-side of the adder is swapped upon gating to SAL; the digit at adder A-side (0–3) is gated to SAL(4–7), and A-side (4–7) is gated to SAL(0–3). This function is used mainly by the Pack and Unpack instructions to interchange the sign and digit positions.
2. Zone or sign insertion. The correct zone or sign code (USASCII-8 or EBCDIC) is applied from the ROSDR to the adder A-side. The zone or sign may be merged with the digit in any combination.
3. Invalid digit and sign detection. The inputs to the A and B sides of the adder are tested for invalid digits or signs. An appropriate interrupt trigger is set upon detection of an invalid code.
4. Zero detection. This function is used to sense overflow conditions and also to detect all-zero results. An all-zero result placed into main storage must carry a positive sign. Consequently, arithmetic instructions such as Add and Subtract specify testing of each SAL byte for zeros. If upon execution of the instruction it is found that an all-zero result has been stored, the instruction forces storing of a plus sign at the low-order byte address.

WORD OVERLAP CONDITION

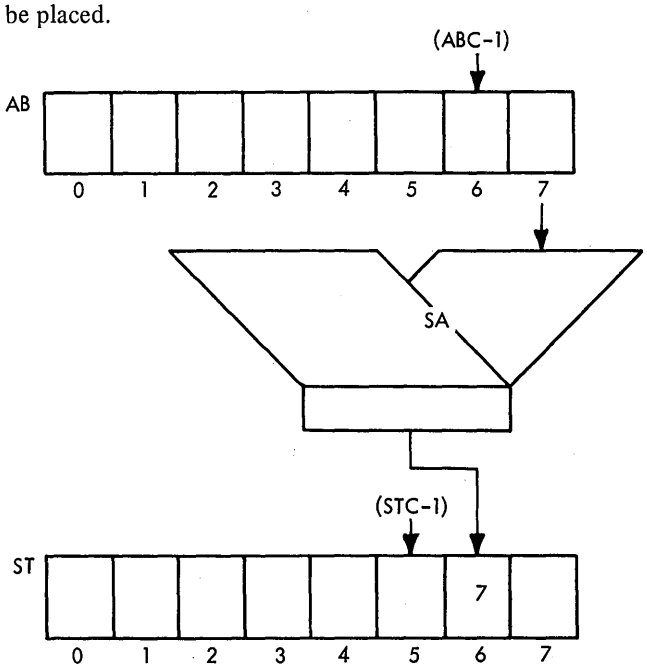
- Word overlap condition exists when: IC(0–20) = D(0–20) and IC(21–23) > D(21–23).
- Test for word overlap is performed by GIS of all one-operand instructions.
- Execution of one-operand instructions provides separate microprogram to handle word overlap conditions.
- No special action is taken to detect word overlap in two-operand instructions.
- Word overlap in two-operand instructions causes data program interruption.

Data is fetched from and placed into main storage one doubleword at a time. However, program compatibility of the CE with smaller models in the System/360 requires that all results placed into main storage must be considered to be stored one byte at a time as they are processed. There are some cases where this compatibility would not be maintained unless special actions were taken. The condition that requires special handling is called "word overlap" and occurs when the fields of the first and second operands specified by the instruction overlap.

The operand addresses and field lengths may be such that one or more bytes in main storage are specified as part of both the first and the second operands. For example, consider the case in which the IC and D specify the same doubleword in storage; the IC specifies byte 7 as the starting second operand byte to be processed in this doubleword, and D specifies byte 6 as the starting first operand address. At least two operand bytes are to be processed.



This doubleword is fetched from main storage and placed into AB and ST. A one-operand instruction, such as Move with Offset, will process the first AB byte (byte 7) in the serial adder and place the result into the designated first operand byte; i.e., byte 6 in ST. Then, ABC and STC are reduced one count to designate the next AB byte to be processed and the ST location into which the results must be placed.



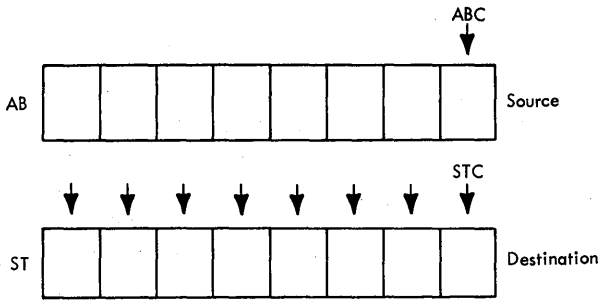
Note, however, that the preceding move operation has replaced the original contents of byte 6 in ST with the contents of byte 7. Thus, the next AB byte to be processed (original byte 6) is no longer valid and must be updated; i.e., the equivalent of storing ST byte 6 and then refetching this byte to AB must be performed.

As seen from the preceding example, the word-overlap condition may require special handling of data. Execution of all one-operand decimal instructions (Pack, Unpack, Move with Offset, and Zero and Add) provides for two

alternate microprograms. One microprogram is for the normal, or not-word-overlap, case; the other handles the word-overlap condition. Selection of the appropriate microprogram is dependent on the outcome of the word-overlap test, which is performed in the General Initialization Sequence (GIS) of all one-operand instructions.

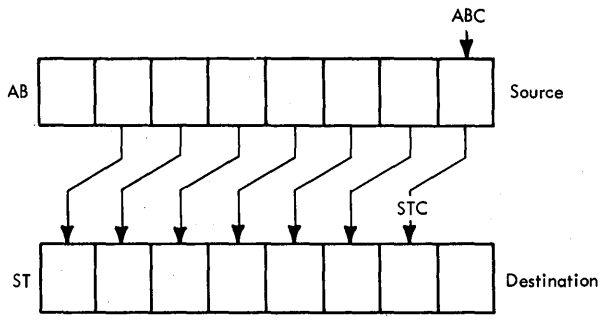
A word-overlap condition exists when both operands have the same doubleword address. The manner in which the first and second operand bytes are specified within this doubleword determines whether special data handling is required. When the word-overlap condition exists, three cases of byte specification may be distinguished:

1. The first and second operand bytes are the same; no special data handling is required.†



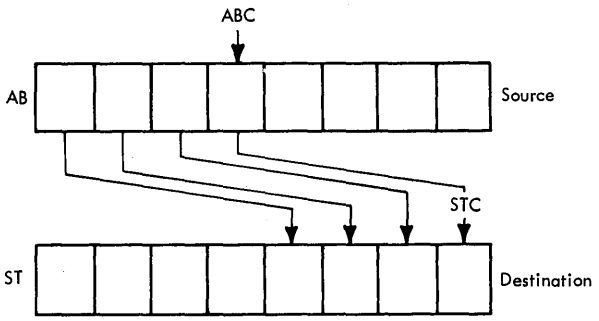
In this case, the destination bytes are placed into the same locations from which the source bytes are obtained. Because processing of any source byte does not affect the contents of the next source byte, no updating of source bytes is necessary.

2. The first operand bytes are specified "ahead" of the second operand bytes; special data handling is required.



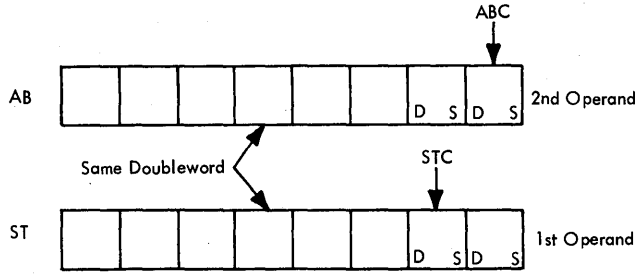
In this case, the destination bytes are placed into the locations from which the next source bytes will be processed. The data in AB becomes "obsolete" after processing of one or more source bytes. (The crossover point at which data becomes obsolete depends on the amount of skew between ABC and STC.) This word overlap case requires special data-handling techniques.

3. The first operand bytes are specified "behind" the second operand bytes; no special data handling is required.



In this case, the destination bytes are placed behind the source bytes as they are processed. Thus processing of any source byte cannot affect the contents of the next source byte, and no updating is necessary.

In two-operand arithmetic instructions, no special action is taken to detect word overlap. Word overlap is ignored during execution of a Compare instruction, because this instruction does not store the results. The operand fields specified for Add, Subtract, Multiply, and Divide instructions either should not overlap at all or should have coincident rightmost bytes. The GIS for these instructions does not perform the word overlap test, because improper overlap of the operands causes an invalid data condition to be detected in the execution phase. In two-operand instructions, the operand fields are correctly specified when the rightmost byte of each operand contains the operand sign; all bytes to the left of the sign byte must contain only digits. This requirement cannot be fulfilled when both operands in the instruction specify the same doubleword with different byte addresses. The following example shows that the sign byte of the first operand is also the "digit" byte of the second operand.



During execution of the instruction, all operand digits are checked for validity. Detection of a sign code in the digit position forces a data program interruption.

† Except for Unpack instruction. This instruction generates two bytes of destination for each byte of source and requires special data handling in all cases of word overlap.

GENERAL INITIALIZATION SEQUENCE

At the completion of the SS I-Fetch, the CPU is in the following status:

1. A main storage request for the doubleword containing the low-order byte of the first operand has been issued per D.
2. D contains the low-order byte address (contents of GPR addressed by B1, + D1, + L1) of the first operand.
3. The IC contains the high-order byte address (contents of GPR addressed by B2, + D2) of the second operand.
4. The next instruction address has been transferred from the IC to the LSWR.
5. E(0–7) contains the instruction op code.
6. The 'PSC' trigger has been set, if appropriate.

Following the SS I-Fetch, a branch is made per the instruction op code to the appropriate General Initialization Sequence (GIS). The general functions of the GIS for decimal instructions are described below. Functions peculiar to a specific instruction are covered in subsequent paragraphs dealing with the execution of that instruction.

The function of the GIS microprogram is to set up initial conditions for the execution phase. These include:

1. Gating the first operand from the SDBO to ST. A D request for the doubleword containing the low-order byte of the first operand was issued during SS I-Fetch.
2. Adding of L2 field to IC. At the end of SS I-Fetch, the IC contains the address of the high-order byte of the second operand. To address the low-order byte of the second operand, the IC must be incremented by the L2 field during GIS.
3. Initiating a storage request per the IC for the second operand. GIS initiates a storage request for the doubleword containing the low-order byte of the second operand.
4. Setting of STC and ABC. The STC is set to the rightmost first operand byte in ST, the byte to be processed first. Because the address of the rightmost byte is specified by D(21–23), the STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by gating IC(21–23) to the ABC.
5. Gating the second operand from the SDBO to AB. An IC request for the second operand is issued during GIS. Subsequently, the GIS gates the second operand from the SDBO to AB.
6. Performing a sign handling function. For add-type instructions, the sign of the result is tentatively set to agree with the sign of the first operand before the execution phase. The GIS examines the signs in the rightmost bytes of both operands and establishes whether to perform a true add or a complement add operation. For multiply and divide operations, the sign of the result is determined during the execution phase by examining the appropriate STAT's, which have been previously set according to the signs of the two operands. Both operands are tested for an invalid sign.
7. Performing the word overlap test. A word overlap test is performed during the GIS for Pack, Unpack, Move with Offset, and Zero and Add instructions.

ADD, SUBTRACT, AND COMPARE

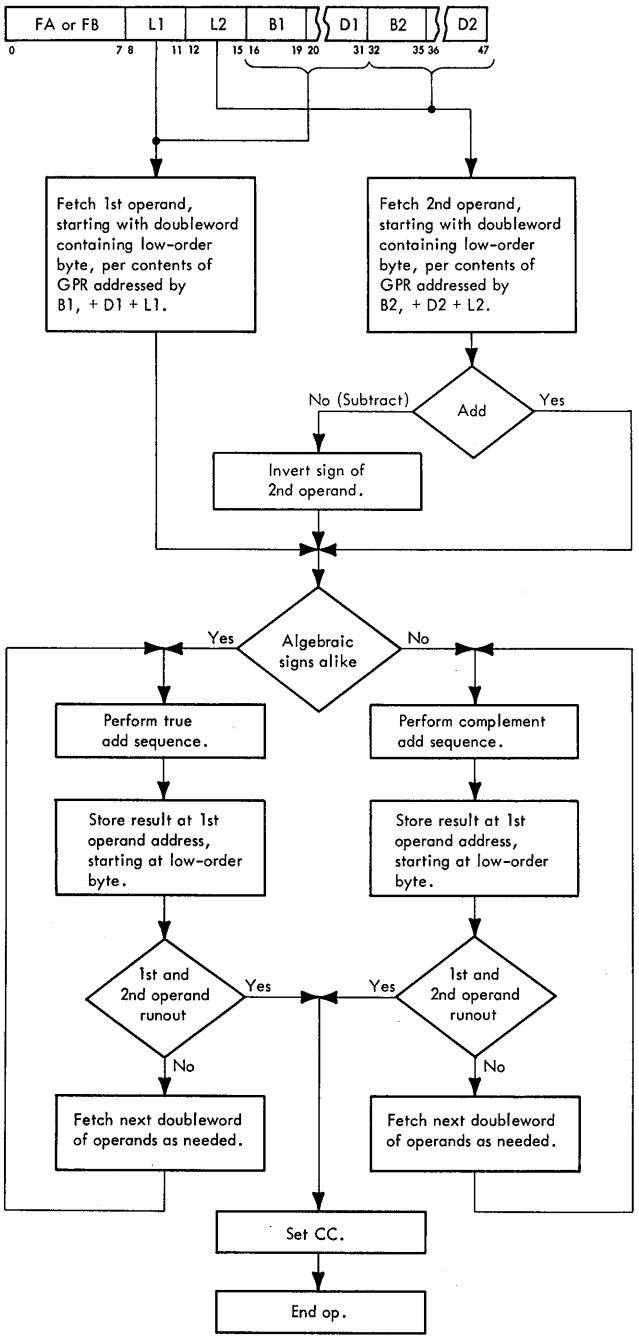
During the GIS, separate sign handling is performed for the Add, Subtract, and Compare instructions; the sign of the second operand is, in effect, inverted for Subtract and Compare instructions. After exit from the GIS, the three instructions share a common true add or complement add routine, depending on the operand signs and the instruction. Because the result of a Compare instruction is not stored into main storage, the setting of the mark triggers and overflow detection are inhibited in hardware during execution of this instruction.

Add, AP (FA) and Subtract, SP (FB)

- Algebraically add (subtract) 2nd operand (in storage) to (from) 1st operand (in storage) and place result into 1st operand location.
- SS format: (See following page.)
- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.
  Overflow: CC = 3.

The Decimal Add and Subtract instructions specify an algebraic addition with the sign of the second operand inverted for the subtract instruction. The sign of the result is tentatively set to agree with the sign of the first operand. Then if the algebraic signs of the two operands are alike, a true add sequence is performed; if unlike, a complement add sequence is performed. The tentative result sign if correct in all cases except for a complement add operation where the magnitude of the second operand is equal to or greater than the magnitude of the first operand. If the magnitudes of the two operands are equal, the result is zero and a positive sign is stored into the result field. If the magnitude of the second operand is greater than that of the first, a complement result is formed and the sign of the result is inverted by setting it to the algebraic sign of the second operand when the result is recomplemented. All signs and digits are tested for validity. The operand fields may overlap when their low-order bytes coincide; therefore, it is possible to add a number to itself.

The result is stored at the first operand address. If the first operand field is too short to contain all significant digits of the result, a decimal overflow occurs.



GIS for Add and Subtract

The GIS microprogram for the Add and Subtract instructions is shown in Diagram 5-301, FEMDM. It performs the following functions:

1. Loads into ST the doubleword containing the low-order byte of the first operand.
2. Adds L2 to the IC, after which it requests, per the IC, the doubleword containing the low-order byte of the second operand. When this doubleword arrives, loads it into AB.
3. Transfers the L1 count to F(0–3).
4. Assigns a result sign that agrees with the sign of the first operand.
5. Performs a branch on the algebraic signs of the operands (contained in STAT's C and F) to enter the true add or complement add sequence.

At the start of GIS, the IC contains the address of the high-order byte of the second operand (contents of GPR addressed by B2, + D2). To obtain the address of the low-order byte, L2 is added to the IC, and a main storage request is issued per the result.

The L1 count in E(8–11) is destroyed during subsequent execution and must be preserved in F(0–3). This action is necessary because, upon execution of the instruction, it may be found that results were placed in main storage in complement form. Because the final result must be true, the destination field is refetched and recomplemented, and the sign is inverted. In such cases, the L1 count in F(0–3) is used to refetch the correct number of destination bytes.

The result is arbitrarily assigned the sign of the first operand by performing a branch of STAT's F and C. STAT F is set if the first operand is minus. Note, however, that the sign of the second operand is not known at this time and STAT C will always be in the reset state. Thus, when STAT's F and C are alike, it indicates that STAT F is not set (first operand plus); when the STAT's are not alike, it indicates that STAT F has been set and the first operand is negative.

A second branch on STAT's F and C is performed after the sign of the second operand has been sensed, and STAT C set accordingly. STAT C is set for a minus sign for an Add instruction and for a plus sign for a Subtract instruction. This is the only difference in the execution of an Add or Subtract instruction. If the STAT's are alike, a true add sequence is entered. Upon entry into this sequence, the result always carries the correct sign. If the STAT's are not alike, a complement add sequence is entered. In this case, the algebraic sign of the result cannot be known at the start of the operation because it is dependent on the relative magnitude of the operands. If the sign has been assigned incorrectly, the result of the complement add operation will be in complement form. This condition will be detected at the completion of the instruction, in which case the result will be recomplemented and the sign inverted by setting the result sign to the algebraic sign of the second operand (per STAT C).

True Add Sequence

- True +6 add operation exits on one or more of the following conditions:
  L1 or STC = 0.
  L2 = 0.
  ABC = 0.

- STAT A is set if result is not zero.
- STAT B is set if overflow occurs.
- STAT E is set if operand digit or sign is invalid.
- STAT G is set if Compare instruction.
- STAT H is set if carry to next byte occurs.

An overall flowchart of the true add sequence and the data path used for its execution are shown in Sheet 1 of Diagram 5-302, FEMDM. The flowchart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into the true add sequence, the signs of both operands have been examined (by the GIS) and the correct sign has been entered into the low-order destination byte in ST. The first step in the microprogram is to true-add the digits contained in the sign bytes of the operands. The result is then placed into the digit portion of the destination byte. At this point, one complete byte of the result has been developed. The operand length codes (L1 and L2) and the status of the byte counters (STC and ABC) are examined for one or more of the following exit conditions from the true add loop:

1. STC and L1 $\neq 0$, L2 = 0.
   The second operand field has run out.
2. STC and L1 $\neq 0$, L2 $\neq 0$, ABC = 0.
   More second operand bytes are needed.
3. STC or L1 = 0, L2 $\neq 0$, ABC = 0.
   The first operand field has run out, or ST is full and more first operand bytes are needed. In either case, ST must be stored into the destination address per D. More second operand bytes are needed.
4. STC or L1 = 0, L2 $\neq 0$, ABC $\neq 0$.
   The same conditions exist as in item 3 except that more second operand bytes are not needed.
5. STC or L1 = 0, L2 = 0.
   The first operand field has run out, or ST is full and more first operand bytes are needed. In either case, ST must be stored into the destination address per D. The second operand field has run out.

If none of the above exit conditions exist, the microprogram re-enters the true add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB and ST bytes are added in the serial adder, and the result replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed, and, if L1 is not zero, the true add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test and a test for all-zero result.† If an all-zero result has been obtained, the address of the low-order destination byte is restored in D and a plus is stored at this address. Restoration is necessary, because D is decremented by 8 for each doubleword of first operand that is fetched. If, for example, two doublewords of first operand have been fetched, the address of the low-order destination byte is obtained by adding 16 to D.

A detailed flowchart of the true add sequence is shown in Sheet 2 of Diagram 5-302. It is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the true add operation. This data handling is straightforward for the most part and requires no explanation. Those areas in need of clarification are discussed in the following subparagraphs.

True Add Operation. The selected AB byte is gated (true +6) to the serial adder and added as a binary number to the selected ST byte. The adder output is decimal-corrected at the input to SAL and gated back to the selected ST byte. For the first (or sign) byte, only bits 0–3 of the selected AB byte are gated to the adder. The decimal correction involves examining the carry from each digit and logically subtracting 6 from each result digit that did not have a carry. As each byte is processed, ABC, STC, L1 and L2 are decremented by 1 and the selected mark trigger is set (except for a Compare instruction). The carry from each byte is saved in STAT H and, if set, results in a carry to bit 7 of the next byte processed. STAT A is set if any nonzero result digit is detected. STAT E is set if any invalid digit is detected at the inputs to the serial adder.††

Exit Conditions. An exit is made when one or more of the following conditions exist as determined by a functional branch micro-order ('Decimal' micro-order):

1. L1 or STC = 0.
2. L2 = 0.
3. ABC = 0.

Five possible exit conditions exist:

1. STC and L1 $\neq 0$, L2 = 0.
   The second operand field has been completely processed. AB and ABC are cleared, and the high-order source extend routine is started to process the remaining destination field.

† The Add, Subtract, and Compare instructions have the same microprogram. The Compare instruction does not store the result into main storage and, upon exit from the true add loop, enters the end-op sequence.

†† When an invalid digit is detected at the serial adder in-buses, the adder forces 1's into its sum and parity output latches. This action insures that valid parity is always gated to ST from the serial adder.

2. STC and L1≠0, L2≠0, ABC = 0.
   A second operand fetch sequence is started to fetch the next doubleword of second operand to AB.
3. STC or L1 = 0, L2≠0, ABC = 0.
   The first operand has either run out, or ST is full and the next doubleword of the first operand is needed. In either case, ST must be stored if an Add or Subtract instruction is being executed. (If a Compare instruction is being executed, no mark triggers are set and ST is not stored.) The next doubleword of the second operand is needed.

   After the destination store cycle, L1 is tested for all 1's to determine whether the L1 field has run out. If L1 equals all 1's, a second operand fetch is initiated before entering the first operand runout sequence; if L1 does not equal all 1's, the first operand fetch sequence is performed, followed by the second operand fetch sequence and resumption of the true add loop.
4. STC or L1 = 0, L2≠0, ABC≠0.
   The same conditions exist as in item 3 except that a second operand fetch is not needed and is not performed.
5. STC or L1 = 0, L2 = 0.
   A storage request per D is issued to store ST into the destination field (unless a Compare instruction is being executed). AB and ABC are cleared to start the high-order source extend routine. A further test is required to determine whether L1 was zero.

   If L1 is now all 1's, all destination bytes have been processed. A carry from the last destination byte indicates an overflow condition, and STAT B is set.

   If L1 is not all 1's, a first operand fetch sequence is started, after which the high-order source extend routine is resumed.

First Operand Fetch. A separate entry is made into this routine, per STAT G, for a Compare instruction. In a compare operation, a D request for the next doubleword of first operand has already been given. D is decremented by 8, and the doubleword arriving at the SDBO is gated to ST.

For Add or Subtract instructions, D is decremented by 8, and a D request is made for the next doubleword of first operand. F is incremented by 1 to record the number of fetches made. This information will be required to restore the low-order address in D in case an all-zero result is obtained. If ABC equals 111, a second operand fetch routine is started. If ABC does not equal 111, the appropriate addition or high-order source extend is started.

Second Operand Fetch. The IC is decremented by 8, and the next doubleword of second operand is fetched to AB. After this, the appropriate addition or overflow routine is started as determined by the L1 count.

Second Operand Runout. An all-zeros AB byte is gated true +6 to the serial adder and added to the selected ST byte. The result is decimal-corrected and gated back to the selected ST byte. STAT's A, E, and H and the mark triggers are set as previously explained.

L1 and STC are decremented by 1 as each byte is processed; ABC and L2 are not stepped. The sequence is repeated until L1 is stepped to zero, with an exit to the destination store and first operand fetch sequences whenever STC equals 7. A carry from the last destination byte is an overflow condition and sets STAT B. The end-op sequence is started when L1 equals zero.

First Operand Runout and Overflow Test. An overflow condition exists whenever a carry results as the last destination byte is processed or whenever a nonzero digit is detected in the source field after the destination field has been processed. STAT B is set if STAT H is set when entering this routine. Next, the remaining second operand bytes are gated true +6 to the serial adder with STAT B being set if any nonzero bytes are detected.

ABC and STC are decremented by 1 as each byte is processed. The next source doubleword is fetched to AB whenever ABC is stepped to zero unless L2 equals zero. When L2 is stepped to zero, the end-op sequence is started.

Zero Result. If at the completion of the true add operation STAT A is not set, an all-zero result has been obtained. In this case, the Add and Subtract instructions always force a positive sign into the low-order byte of the destination field. (If STAT G is set, an exit is made to the end-op sequence because no correction of the result is required for a Compare instruction.)

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4–7). STC is set per D(21–23), and the selected ST byte is cleared.

STAT B is examined to determine an overflow condition. For a zero result and no overflow, a plus sign is inserted via the serial adder into the low-order destination byte with the selected mark trigger being set. A storage request is given to store the sign into the destination field, and the end-op sequence is started. For a zero result and overflow, the destination sign is not corrected. The end-op sequence is started immediately.

End-Op Sequence. The instruction address (original content of the IC) is restored from the LSWR to the IC, and STAT G is reset. A data program interruption occurs if STAT E is set. A decimal overflow program interruption occurs if STAT B is set and STAT E is not set. The CC is set per hardware conditions as shown in Table 3-12.

Table 3-12. Condition Code Setting Per Hardware Conditions, Decimal Instructions

Hardware Conditions	Setting
Not STAT B • Not STAT A • (Add, or Subtract, or Zero and Add)	0
Not STAT A o Not STAT H o Compare	0
Not STAT A • STAT H • STAT F o Not STAT C o Compare	0
Not STAT A • STAT H • Not STAT F • STAT C • Compare	0
Not STAT B • STAT A • STAT F • (Add, or Subtract, or Zero and Add)	1
STAT A • STAT F • STAT H • Compare	1
STAT A • STAT C • Not STAT H • Compare	1
STAT F • STAT C • STAT H • Compare	1
Not STAT B • STAT A • Not STAT F • (Add, or Subtract, or Zero and Add)	2
STAT A • Not STAT F • STAT H • Compare	2
STAT A • Not STAT C • Not STAT H • Compare	2
STAT H • Not STAT F • Not STAT C • Compare	2
STAT B • (Add, or Subtract, or Zero and Add)	3

Note: • Designates logical AND connective.

Complement Add Sequence

- Complement add operation with exits on one or more of the following conditions:
  1. L1 or STC = 0.
  2. L2 = 0.
  3. ABC = 0.
- STAT A is set if result is not zero.
- STAT B is set if overflow occurs.
- STAT D is set if result must be recomplemented.
- STAT E is set if operand digit or sign is invalid.
- STAT G is set if Compare instruction.
- STAT H is set if carry to next byte occurs.
- Carry out of last destination byte indicates result is in true form; no carry condition indicates result is in complement form.

Diagram 5-303, Sheet 1, FEMDM, is an overall flowchart of the complement add sequence. This flowchart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into this sequence, the signs of both operands have been examined (by the GIS) and the sign of the first operand has been inserted as the sign of the result. This sign may or may not turn out to be the correct sign: if the first operand is larger than the second, the result carries the correct sign; if the reverse is true, the sign of the result must be inverted.

Basically, the complement add microprogram is similar to the true add sequence previously described. The first step in the microprogram is to complement add the digits contained in the sign bytes of the operands. The result is then placed into the digit portion of the destination sign byte. At this point, one complete byte of result has been developed. The operand length codes (L1 and L2) and the status of the byte counters (STC and ABC) are examined for one or more of the following conditions:

1. The result byte is contained in the last byte of ST and must be stored (STC = 0).
2. Additional first operand bytes must be fetched from main storage (STC = 0 and L1 $\neq$ 0).
3. Additional second operand bytes must be fetched from main storage (ABC = 0 and L2 $\neq$ 0).
4. The second operand has run out; i.e., all second operand bytes have been processed, and zeros must be added to the first operand bytes (L2 = 0 but L1 $\neq$ 0).
5. The first operand has run out; i.e., the destination field has been completely processed (L1 = 0).

If none of the above conditions exist, the microprogram enters the complement add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB byte is complement-added to the selected ST byte, and the result replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed and, if L1 is not zero, the complement add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test, a zero-result test, and a complement result test.† If an

† The Compare instruction does not store the result into main storage and, upon exit from the subtract loop, enters the end-op sequence.

all-zero or complement result has been generated, the address of the low-order destination byte is restored to D. Then, the result is either recomplemented and the sign inverted or a plus is stored into the low-order destination byte.

Sheet 2 of Diagram 5-303 is a detailed flowchart of the complement add sequence. It is an expanded version of the overall flowchart, showing the data handling in the various subroutines of the complement add operation. The complement add operation is similar to the true add sequence described previously. For this reason, only the differences are discussed below.

Complement Add Operation. The selected AB byte is converted to 2's complement form at the input to the serial adder, and is then added to the selected ST byte. For the first (or sign) byte, bits 0–3 only of the selected AB byte are gated in inverted binary form to the adder, with a hot carry supplied to bit 3 to convert to 2's complement.

Second Operand Runout. An all-zeros AB byte is gated in complement form to the serial adder and added to the selected ST byte. Thus the second operand is extended with high-order binary 1's (decimal 9's + 6).

Overflow Test. Generally, an overflow condition exists, if, upon processing all destination bytes, a non-zero source byte is detected. One exception occurs when the first source byte sensed, after the first operand has run out, equals 1 and a "carry" condition exists. A carry condition is determined by STAT's A and H being set; i.e., a nonzero result and a carry from the previous byte.

When L2 has been stepped to zero, a carry from the last destination byte is examined. A carry condition indicates a true result, and the end-op sequence is started. No carry indicates a complement result, and a recomplement sequence is started for Add and Subtract instructions. For a Compare instruction (STAT G set), the end-op sequence is started immediately.

Zero Result and Recomplement Setup. STAT D is set when an entry is made to this routine because of the result's being in complement form. STAT D is not set when an entry is made because of a zero result. If STAT G is set when entering this routine, an exit is made to the end-op sequence because no corrections of the result are required for the Compare instruction.

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4–7). STC is set per D(21–23), and the selected ST byte is cleared. If an overflow condition exists (STAT B set), the results need not be corrected and the end-op sequence is started immediately.

Recomplement Sequence. The original L1 count was saved in F(0–3) by the GIS. This count is now placed into the L2 location in E; i.e., E(12–15). The L1 location in E(8–11) is set to zero and then decremented one count to provide an exit from the first operand fetch routine to the recomplement sequence. The complement result is gated from the SDBO to AB, and the recomplement sequence is started.

The sign byte is processed by gating bits 0–3 of the selected AB byte in inverted binary form to the serial adder, with a hot carry supplied to bit 3. The sign is inverted by inserting the algebraic sign of the second operand into serial adder bits 4–7, as determined by STAT C. Bits 0–3 are decimal-corrected, and the adder output is gated to ST.

All bytes following the sign byte are processed by gating the selected AB byte complement to the serial adder, where it is added to an all-zero ST byte. The adder output is decimal-corrected and gated back to the selected ST byte.

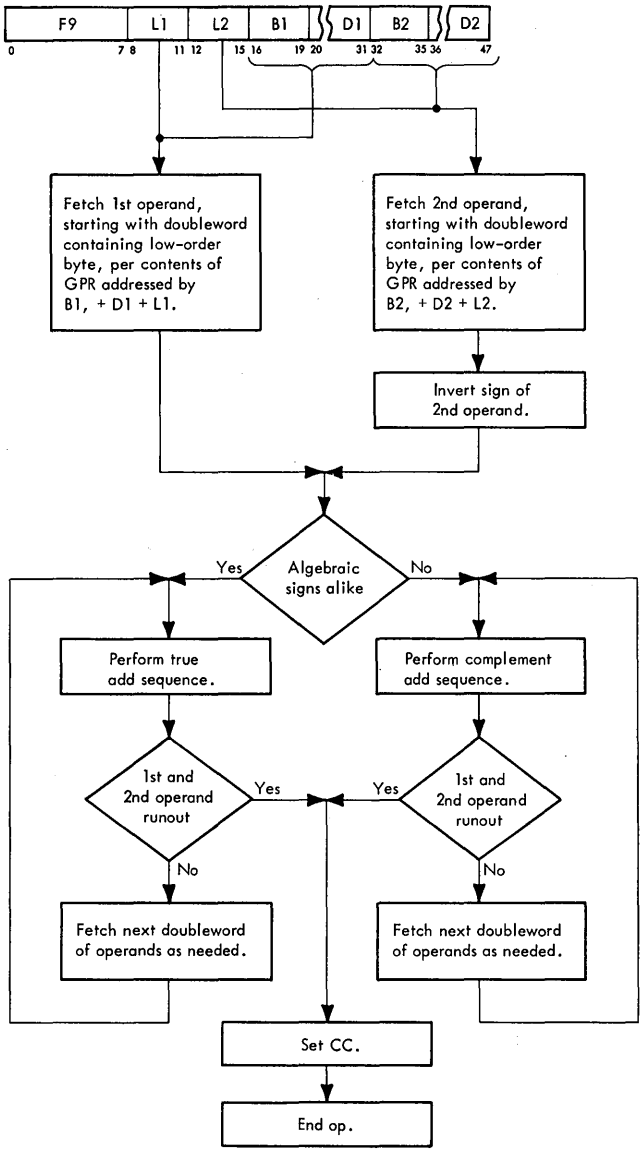
As each byte (including the sign byte) is processed, the ABC, STC, and L2 counts are decremented. The mark trigger selected by the STC is set. The serial adder carry is saved in STAT H. STAT A is set on nonzero digits, and STAT E is set on invalid digits.

Recomplementation is continued until the L1 in E(12–15) is stepped to zero. If ABC steps to zero and L1 is not zero, ST is stored and the next doubleword of destination is fetched to AB. When L1 steps to zero, ST is stored into the destination field, AB is cleared, and STAT F is set or reset per STAT C. The CC is set per hardware conditions (see Table 3-12), and the instruction is ended.

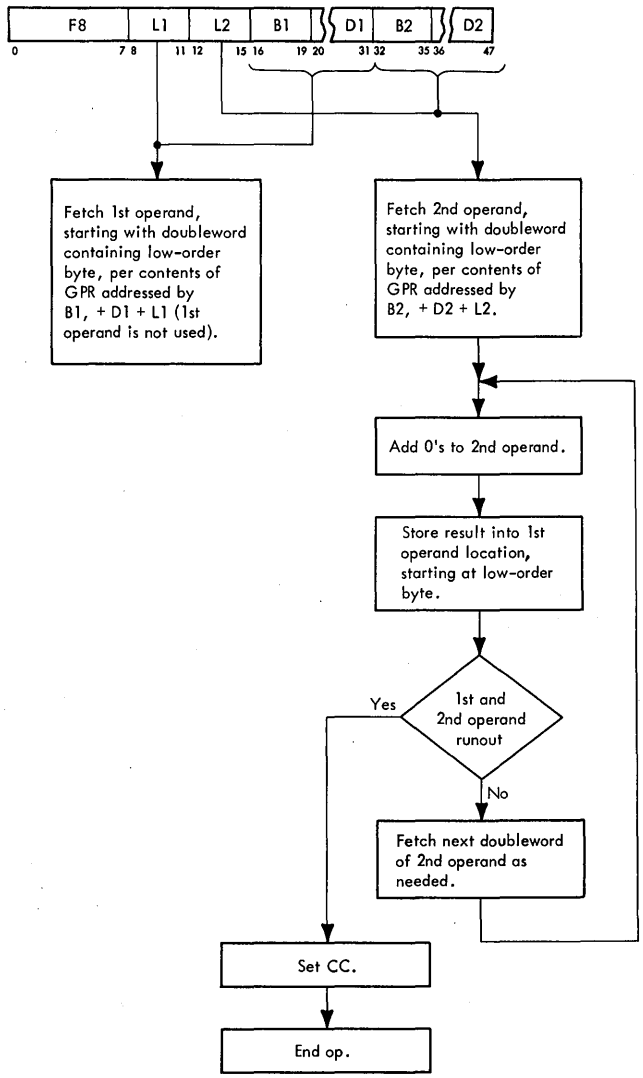
Compare, CP (F9)

- Algebraically compare 1st operand (in storage) with 2nd operand (in storage) and set CC according to result.
- SS format: (See left column of next page.)
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The CP instruction shares the same true add and complement add routines used by the Add and Subtract instructions. The GIS microprogram for the Compare instruction is shown in Diagram 5-301. As in the Subtract instruction, this microprogram effectively inverts the sign of the second operand by setting STAT C on a positive sign.



- SS format:



The result of the compare operation is not placed into main storage. STAT G is set to provide a means of taking special action, where required for the Compare instruction, during execution of the common true add or complement add sequences.

- CC setting:
  2nd operand is zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.
  2nd operand cannot fit into destination field: CC = 3.

ZERO AND ADD, ZAP (F8)

- Place 2nd operand (in storage) into 1st operand location (in storage).

The operation specified by the ZAP instruction is equivalent to addition to zero. A zero result is always made positive. When high-order digits are lost because of overflow, a zero result has the sign of the second operand.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied if needed. When the first operand field is too short to contain all significant digits of the second operand, a decimal overflow occurs and results in a decimal overflow program interruption, provided that the decimal overflow mask bit, PSW(37), is 1. The first and second operand fields may overlap when the rightmost byte of the first operand field is coincident with or to the right of the rightmost byte of the second operand. A flowchart of the GIS and execution of the Zero and Add instruction is shown in Diagram 5-304, FEMDM.

At the start of the GIS, the following actions have been performed by SS I-Fetch: (1) a D request has been issued for the doubleword containing the low-order byte of the first operand; (2) the low-order byte address of the first operand (contents of GPR addressed by B1, + D1 + L1) is in D; and (3) the high-order byte address of the second operand (contents of GPR addressed by B2, +D2) is in the IC. During the GIS, the doubleword containing the low-order byte of the first operand is gated from the SDBO to ST, the IC is incremented by the L2 count to address the low-order byte of the second operand, the STC is set to the low-order destination byte, an IC request for the second operand is issued, and a word overlap test is performed. If a word-overlap condition is predicted by this test, the instruction address is restored to the IC, the 'invalid data interrupt' trigger is set, and the instruction is ended. If no word-overlap condition is detected, the doubleword containing the low-order byte of the second operand is gated from the SDBO to AB.

The sign byte is processed by gating bits 0–3 of the selected AB byte to the serial adder. Bits 4–7 of the AB byte are decoded for a positive, negative, or invalid sign. The approved plus or minus sign is inserted into SAL(4–7) and gated, with the digit, to the selected ST byte.

All bytes following the sign byte are processed by gating the selected AB byte true +6 to the serial adder. The selected ST byte is not gated to the adder, and the validity check at the adder B-side is inhibited in hardware. The adder output is decimal-corrected and gated to the selected ST byte.

As each byte is processed, including the first byte, the ABC, STC, L1, and L2 are decremented, the selected mark trigger is set, STAT E is set for invalid data, and STAT A is set for a nonzero digit.

The byte-by-byte transfer from AB to ST is continued until one or more of the following exit conditions are detected via a ROS branch ('Decimal' micro-order):

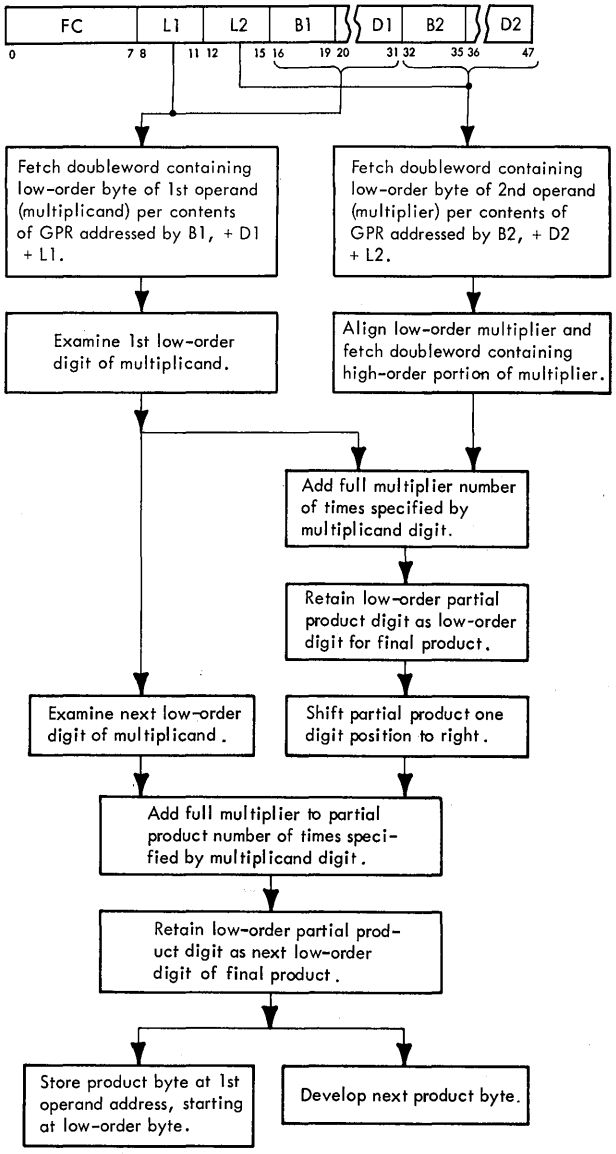
1. STC and L1 $\neq 0$, L2 = 0.
   The second operand field has run out. AB and ABC are cleared and zeros are gated to ST per the STC until L1 or STC equals zero. If the STC is reduced to zero before L1, ST is stored per the D address, and zeros are gated to ST per the STC until L1 is reduced to zero.
2. STC and L1 $\neq 0$, L2 $\neq 0$, ABC = 0.
   More second operand bytes are needed. The next doubleword of the second operand is gated to AB and the zero and add loop is resumed.
3. STC or L1 = 0, L2 $\neq 0$, ABC = 0.
   The destination field has run out or ST is full, requiring a destination store cycle. More second operand bytes are needed. ST is stored into the destination address per D, and the next doubleword of the second operand is fetched per the IC and gated to AB. If L1 equals all 1's (the destination field has run out), AB is tested for nonzero digits to determine if a decimal overflow condition exists. If L1 does not equal all 1's, the zero and add loop is resumed.
4. STC or L1 = 0, L2 $\neq 0$, ABC $\neq 0$.
   The same conditions exist as in item 3, except that a second operand fetch is not needed and is not performed.
5. STC or L1 = 0, L2 = 0.
   The destination field has run out or ST is full, requiring a destination store cycle. The second operand field has run out. ST is stored in the destination address per D. If L1 equals all 1's, both operands have run out. If L1 does not equal all 1's, the second operand runout sequence is performed.

MULTIPLY, MP (FC)

- Multiply 1st operand (in storage) by 2nd operand (in storage) and place result into 1st operand location.
- SS format: (See following page.)
- Maximum multiplicand field (1st operand) is 16 bytes.
- Maximum multiplier field (2nd operand) is 8 bytes.
- Multiplicand field initially contains high-order zero field equal in length to multiplier field.
- L2 $>$ 7 or L2 $\geq$ L1 causes specification program interruption.
- Multiplication accomplished by repetitive addition or subtraction:

Multiplicand Digit	Sequence Selected
0	–
1–4	Addition
5–9	Subtraction

The Decimal Multiply instruction replaces the multiplicand (1st operand) with the product of the multiplicand and the multiplier (2nd operand). To be able to store the product in



the multiplicand field at all times, several restrictions are imposed on both the multiplicand and the multiplier:

1. In any multiply operation, the maximum number of product digits that can be obtained is equal to the sum of the digits in the two operands. Because the product is stored in the multiplicand field, this field must initially contain high-order zero digits for at least a field size equal to that of the multiplier. Thus the multiplicand field is initially split into two parts; the high-order zero field of length equal to the multiplier, and the low-order field containing the effective multiplicand digits. This arrangement of the multiplicand ensures that product overflow will not occur (Figure 3-14).
2. By definition, the multiplier field must be at least one digit less than the multiplicand. Because the multiplicand must initially contain a zero field equal in size to the multiplier digits, the multiplier size is limited to 8 bytes (15 digits and sign). A specification program interruption occurs if the multiplier length code designates more than 8 bytes (L2 is greater than 7), or if L2 is greater than or equal to L1.
3. The maximum product size is 31 digits and sign (16 multiplicand digits plus 15 multiplier digits). The sign is determined algebraically from the multiplier and the multiplicand signs, even if one or both operands are zero. Because during sign resolution two sign positions are merged into one, at least one high-order digit of the product field is zero.

The multiply operation is executed in much the same manner as in manual arithmetic.† The multiplicand is examined one digit at a time, starting with the low-order digit, and the entire multiplier is added the number of times specified by the multiplicand digit. After the first multiplicand digit has been processed, the low-order digit of the resulting partial product (PP) is saved as the low-order product digit. The PP is then shifted one digit position to the right and brought into computation of the next product digit (one order higher than before). This time, the multiplier is added to the PP the number of times specified by the next digit of the multiplicand, and the low-order digit of the new PP thus formed becomes the next product digit. The PP is again shifted to the right, and the sequence is continued until all digits of the multiplicand have been processed. The PP resulting after the last multiplicand digit has been processed becomes the high-order product.

Figure 3-15 illustrates a typical repetitive addition sequence used for multiplication. As each multiplicand byte is processed, the multiplicand length code (L1) is reduced by one count and compared with the multiplier length code (L2). When L1 = L2, all effective multiplicand digits have been processed and the operation is completed.

To reduce the number of computations in the multiply operation, either a repetitive add or a repetitive subtract sequence may be performed. Selection of the sequence is dependent on the magnitude of the multiplicand digit under consideration. An add sequence is selected if the magnitude of the digit is in the range of 1 through 4. For multiplicand digits of magnitude 5 or greater, a subtract sequence is selected. This sequence deducts the multiplier

† The major difference is that the roles of the multiplicand and the multiplier are reversed. Because of its size (up to 16 bytes), the entire multiplicand cannot be held in the CE at one time. For this reason, the full multiplier (up to 8 bytes) is fetched to the CE and multiplied by the individual digits of the multiplicand, which is fetched from main storage 1 byte at a time.

from the PP the number of times specified by the 10's complement of the multiplicand digit and then adds 1 to the next digit of the multiplicand; increasing the next high-order digit of the multiplicand has the effect of adding the multiplier 10 times. For example, the equivalent of a multiplication by 7 is subtracting the operand 3 times to obtain a negative PP and then effectively adding the operand to the PP 10 times.

An example of a typical subtract sequence used for multiplication is shown in Figure 3-16. Note that the PP resulting from a subtract operation is in 10's complement form. When the 10's complement PP is shifted right, its high-order digit position must be extended with a 9.

Following are general and detailed descriptions of the multiply microprogram. The general description outlines the overall structure of the microprogram, enumerates its major functional steps and sequences, and explains their relationship to the overall operation. The detailed description analyzes each sequence individually, making specific references to the register-to-register data transfer in the CE.

General Description

Upon entering the multiply microprogram, the following actions have been performed by SS I-Fetch:

1. A D request has been issued for the doubleword containing the low-order multiplicand byte.
2. The low-order multiplicand address has been placed into D.
3. The IC contents have been transferred to the LSWR, and the high-order multiplier address has been placed into the IC.

An overall flowchart of the multiply microprogram and the general data path used for its execution are shown in Sheet 1, of Diagram 5-305, FEMDM. The major subroutines and functional steps, shown in the figure, are explained below. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do not show the gates and data paths used in the CE, but are intended solely to convey how the multiply algorithm is implemented. For purposes of illustration, a seven-byte multiplicand and a four-byte multiplier are assumed in these diagrams.

General Initialization Sequence

This sequence gates the multiplicand from the SDBO to ST and sets the STC to the low-order multiplicand byte. It increments the address in the IC to the address of the low-order byte of the multiplier by adding L2 to the IC contents. An IC request is issued for the multiplier (second operand), starting at the low-order address. The contents of D are transferred to the STC.

The GIS gates the multiplier from the SDBO to AB and sets the ABC to the low-order multiplier byte. It also performs several actions relating to the subsequent left-adjust sequence of the multiplier:

1. The low-order digit of the multiplicand (in ST) is transferred to F(0–3).
2. STAT F is set if the sign of the multiplicand is negative and STAT E is set if the sign is invalid.
3. The multiplier length code, L2, is transferred to F(4–7).

The functions performed by the GIS are illustrated in Figure 3-17.

Specification Test

This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies eight bytes or less and is smaller than L1.

Incorrect Specification

Detection of an invalid specification forces a specification program interruption. The instruction address is restored from the IC to the LSWR, and the instruction is ended.

Multiplier Left-Adjust Sequence

The multiplier bytes are transferred from AB to ST in such a manner that the high-order multiplier byte occupies the leftmost byte in ST. STAT C is set if the multiplier sign is negative and STAT E is set if the multiplier sign is invalid.

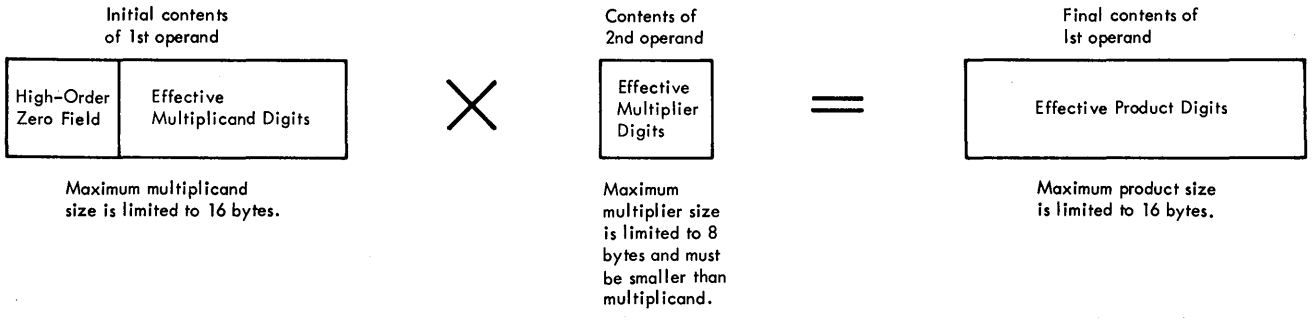


Figure 3-14. Operand Specifications for Decimal Multiply Instruction

Multiply (+204) by (-32) to obtain a product of (-6,528).

Execution:

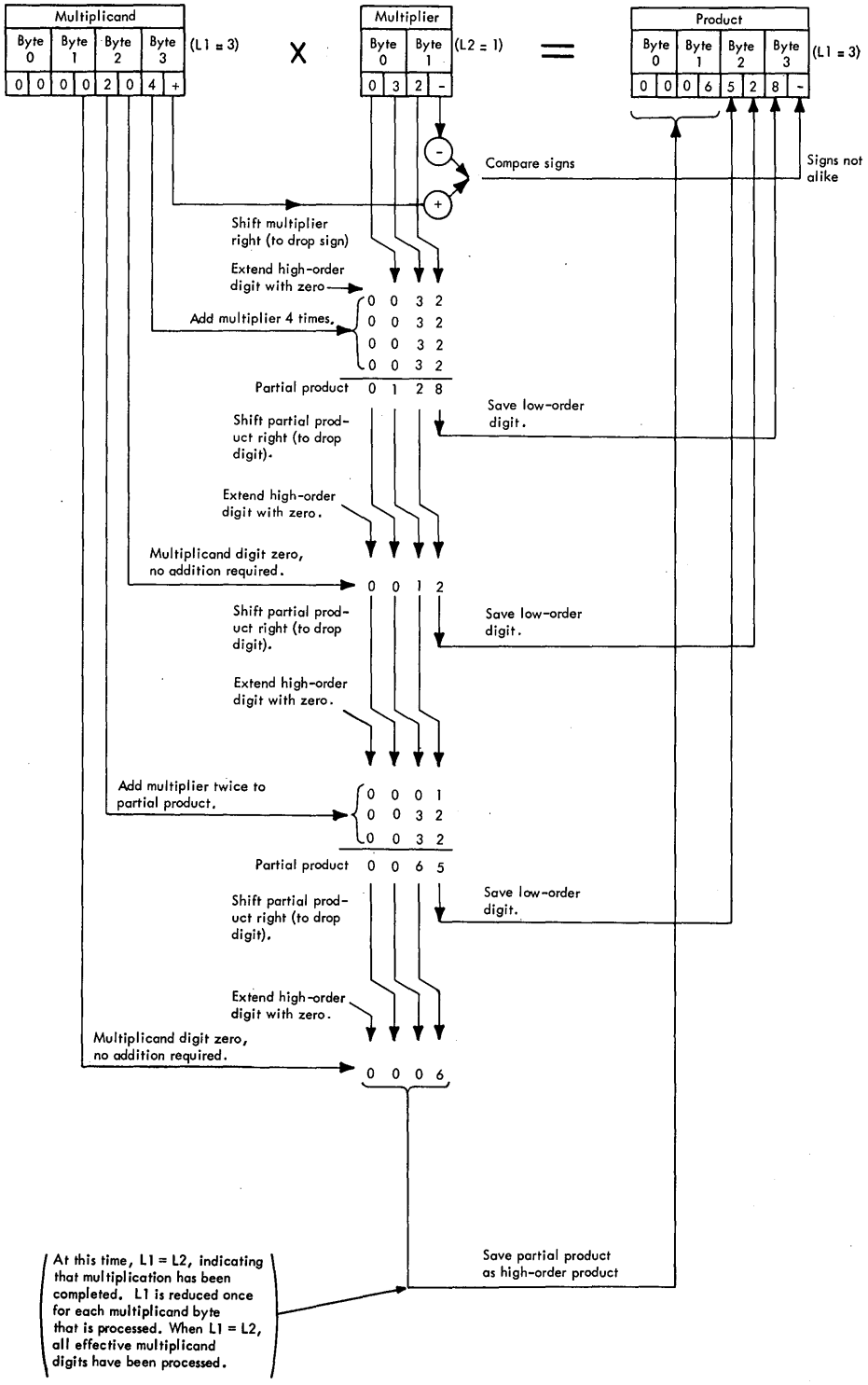


Figure 3-15. Typical Multiply Add Sequence

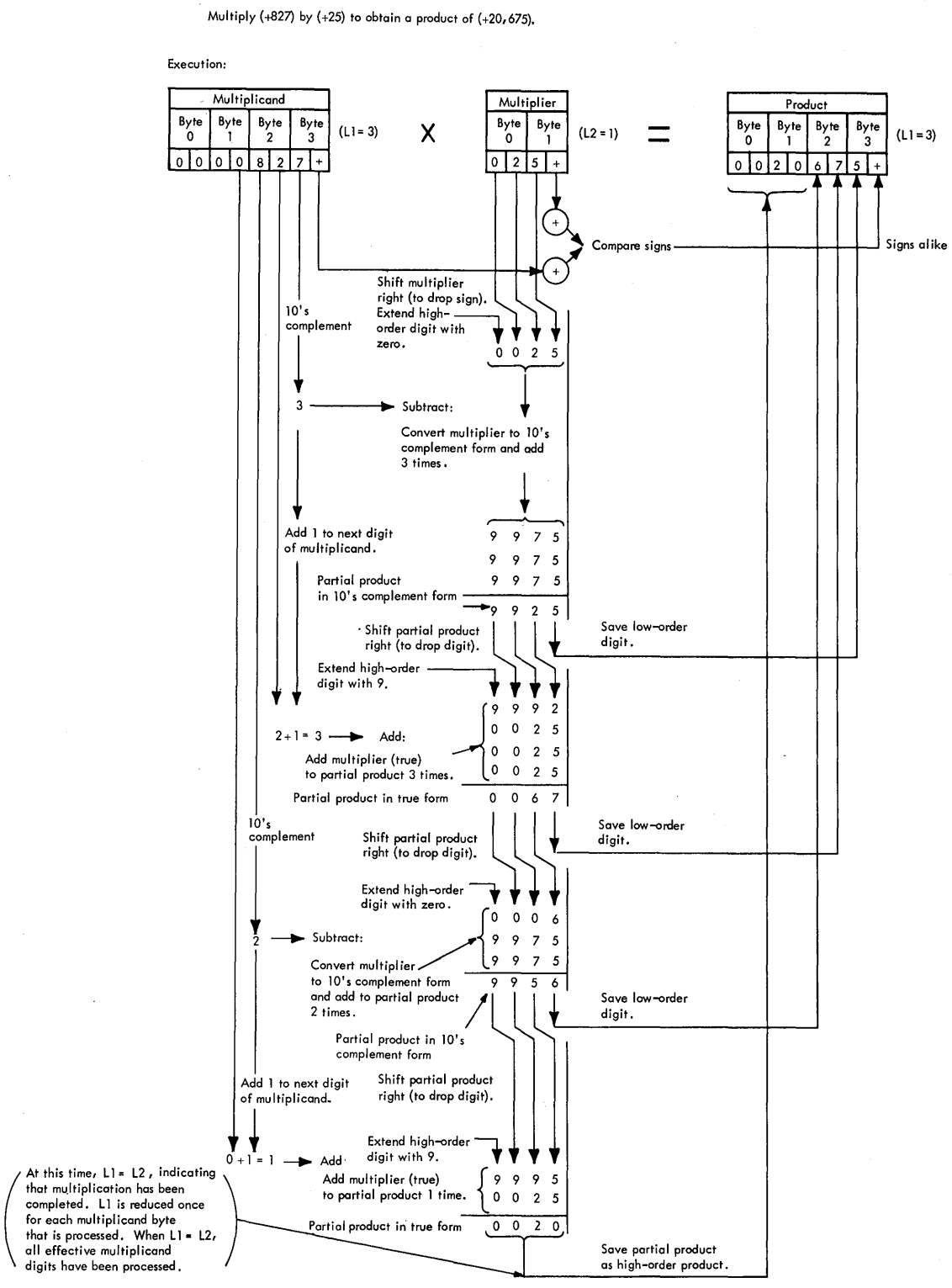


Figure 3-16. Typical Multiply Subtract Sequence

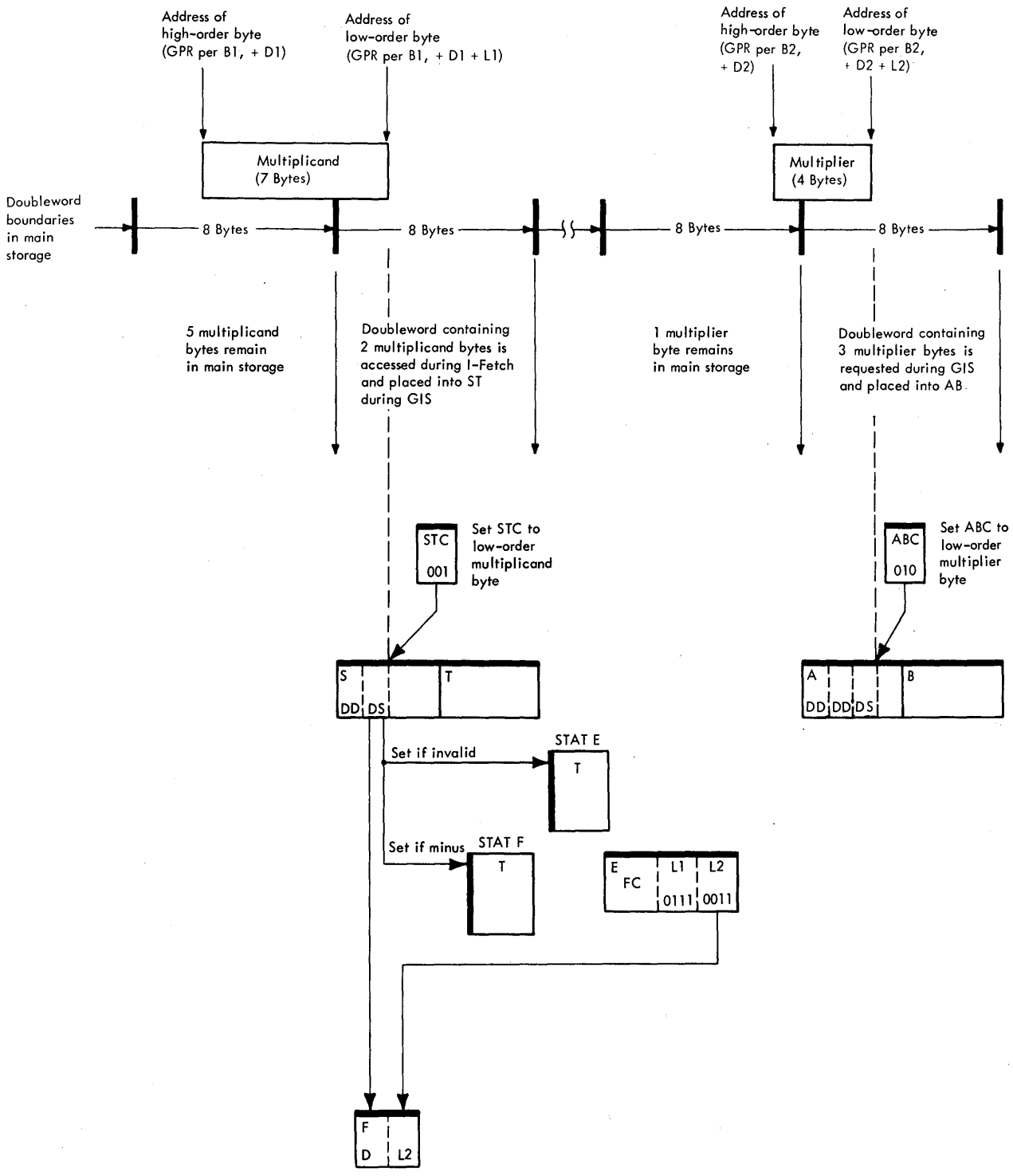


Figure 3-17. Data Handling During GIS of Decimal Multiply

The left-adjust transfer is initiated by setting the STC per L2 (Figure 3-18). Because the maximum multiplier length is limited to eight bytes, only three of the four bit positions in L2 are needed to specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for 1 byte) to a maximum of 0111 (for eight bytes). Setting the STC per L2 automatically selects, according to the multiplier size, the correct ST position for the low-order byte of the multiplier; the number of bytes to the left of the selected ST position corresponds to the length field of the full multiplier.

The transfer is performed one byte at a time, through the serial adder, starting with the low-order multiplier byte. ABC, STC, and L2 are decremented by 1 for each byte transferred. The multiplier is completely transferred when the L2 count is decremented to zero. Because the first IC request (during I-Fetch) does not necessarily gate the full multiplier to AB, it may be necessary to fetch the balance of the multiplier from main storage. (This fetch occurs if the ABC steps to zero before L2 steps to zero.)

After exit from the left-adjust sequence, the full multiplier has been fetched and left-adjusted to ST. Note that the original ST contents (multiplicand) have been destroyed except for the low-order multiplicand byte, which is saved during the GIS; i.e., digit placed into F(0–3) and sign-recorded by STAT F. The destroyed multiplicand bytes are later refetched from main storage, one byte at a time, as required by the multiply operation.

L2 Restoration

During transfer of the multiplier bytes from AB to ST, the L2 count in E(12–15) is decremented by 1 for each byte transferred. At the completion of the left-adjust sequence, L2 has been decremented to zero. The initial L2 count, saved in F(4–7) during the GIS, is now restored to E(12–15). The L2 count will be required by the subsequent multiply sequence.

Multiplier Right-4 Shift to Drop Sign

In the multiply operation to follow, product bytes are developed by adding the entire multiplier the number of times specified by successive digits of the multiplicand. The sign of the multiplier does not enter into the repetitive

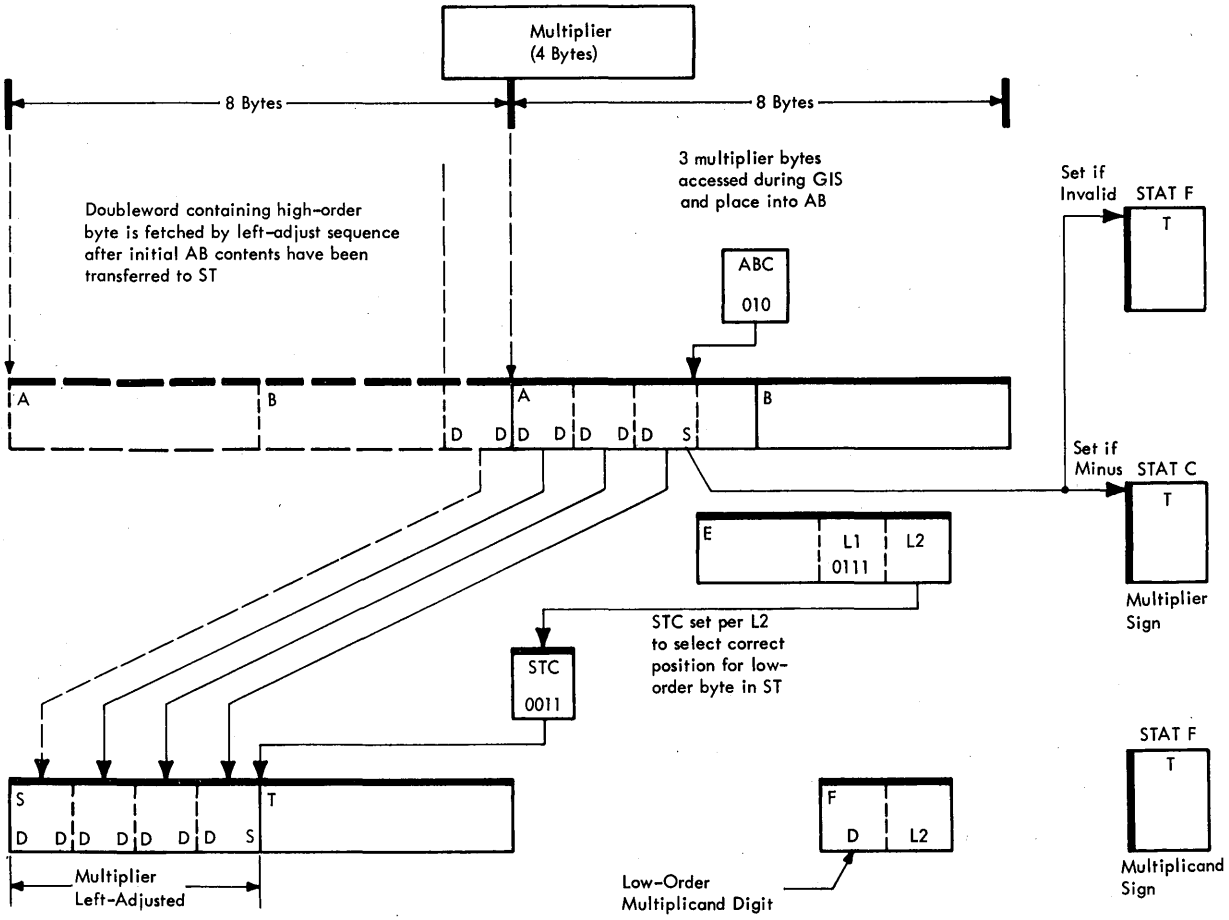
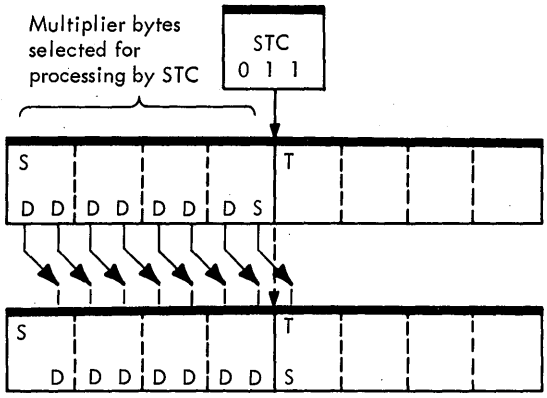


Figure 3-18. Data Handling During Multiplier Left-Adjust Sequence

addition sequence and must be discarded. The sign is discarded by shifting the multiplier in ST 4 bit positions (one digit) to the right as illustrated below. This action places the sign beyond the rightmost multiplier byte selected by the STC for subsequent computation.



Sign Handling

A test of STAT F and STAT C is made to establish the product sign algebraically:

1. Signs alike (both STATS set or reset) – set sign plus.
2. Signs not alike (one STAT set and the other reset) – set sign minus.

Upon establishing the correct product sign, it is placed into F(4–7).

Basic Multiply Add or Subtract Sequence for Left Digit

This sequence processes the digit in the left portion of the multiplicand byte. (The low-order byte of the multiplicand always contains the digit in the left portion and the sign in the right portion.) The entire multiplier (in ST) is added or subtracted the number of times specified by the left digit of the multiplicand saved in F(0–3). An add sequence is performed if the digit in F(0–3) is 4 or less; a subtract sequence, if 5 or greater. A data program interruption occurs prior to a storage cycle if an invalid multiplicand or multiplier digit or sign is detected. The PP resulting from the add or subtract sequence replaces the multiplicand in ST.

Product Byte Store

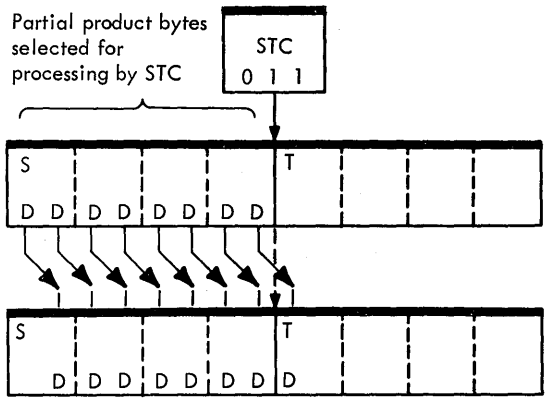
The product is stored into main storage one byte at a time. After exit from the left-digit sequence, one complete byte of product has been developed and must be stored. If the exit is made for the first time, this byte consists of the product sign (in F) and the low-order digit of PP (in ST). All product bytes generated thereafter consist of two digits: one (in F) has been saved from a previous PP developed in the right digit sequence, and the second is the low-order digit of a new PP (in ST) obtained in the left-digit sequence.

Multiplicand Request

A request from D is issued for the next byte of the multiplicand in main storage.

Partial Product Right-4 Shift to Drop Digit

The low-order digit of PP has been stored as a product digit and must not enter into subsequent computation. The digit is discarded by shifting the PP in ST four bit positions to the right. This action places the digit beyond the rightmost PP byte selected by the STC for computation of the next product digit.



L1 = L2

This test establishes whether all digits of the multiplicand have been processed. At the start of the multiply operation, the total field length specified by L1 includes a zero field equal in size to the multiplier plus the effective field of the multiplicand:

L1 = L2 + number of effective multiplicand bytes.

Because L1 is decremented once after each effective multiplicand byte is processed, all the effective multiplicand bytes have been processed when L1 equals L2.

Complete Multiplicand Byte Fetch

If L1 is not equal to L2, the multiply sequence is continued. The next byte of the multiplicand (requested earlier) is selected from the SDBO and placed into F. Control is then transferred to the add or subtract sequence for the right digit of the multiplicand.

Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence

This sequence processes the digit in the right portion of the multiplicand byte. The entire multiplier is added to (or

subtracted from) the PP in ST. The number of add or subtract operations is controlled by the right digit of the multiplicand contained in F(4–7). After a new PP has been developed in ST, its low-order digit replaces the right digit of the multiplicand in F(4–7). The PP is then shifted four bit positions to the right to drop the low-order digit, and an entry is made to the left-digit sequence to process the next multiplicand digit contained in F(0–3).

Multiplicand Zero Test and Partial Product Store

When L1 equals L2, all the effective digits of the multiplicand have been processed. The remaining multiplicand bytes are fetched from main storage and tested for zero. Detection of a nonzero digit results in an interruption. After the zero test is completed, the PP is stored as the high-order product into main storage and the instruction is ended.

Detailed Description

- STAT E is set if digit or sign is invalid.
- STAT A is set if digit is not zero.
- STAT G is set if multiplier is zero.
- STAT H is set to generate hot carry.
- STAT D is set to add 1 to next digit.

Sheet 2 of Diagram 5-305 is a detailed flowchart of the multiply microprogram. This flowchart is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the Multiply instruction. For the most part, this data handling is straightforward and requires no explanation. Those areas in need of clarification are discussed in the following subparagraphs.

General Initialization Sequence

This sequence shares a common microprogram with the Divide instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

Multiplier Left-Adjust Sequence

The ABC has been set to select the low-order multiplier byte in AB. The STC is now set per L2, E(12–15). The transfer is performed one byte at a time via the serial adder. As each byte is gated to the serial adder, it is tested for nonzero value and for invalid digits. STAT E is set upon detection of an invalid digit or sign, and STAT A is set upon detection of a nonzero digit. If upon completion of the left-adjust transfer STAT A remains reset, the multiplier value is zero.

Multiplier Right-4 Shift and L2 Restoration

The multiplier is shifted four bit positions to the right and transferred from ST to AB. L2 is transferred from F(4–7) to E(12–15). Both actions are performed in parallel. As the high-order multiplier bytes are gated from S to PAA and the right-4 shift is initiated, the L2 count is transferred from F(4–7), via the serial adder, to S(28–31). After the right-4 shift has been performed through the parallel adder, the L2 count is gated from S to PAA and the zero count in E(12–15) is gated to PAB. The net result (original L2 count) is gated from PAL to E(12–15).

Sign Handling

STAT G is set if STAT A has not been set during the preceding sequence. This step is taken to indicate a zero multiplier condition which requires special action.

Basic Multiply Add or Subtract Sequence

To perform a branch on the value of the multiplicand digit, the digit must be in SAL(4–7). This requirement is dictated by the 'W=(1–15)' micro-order which samples SAL(4–7). For this reason, the contents of F are cross-gated through the serial adder and placed back into F. SAL(4–7) is then examined for the following values:

1. SAL(4–7) = 0
   No addition cycles are required.
2. SAL(4–7) = 1 through 4
   The multiplier in AB is added to the PP in ST the number of times specified by the digit value.
3. SAL(4–7) = 5 through 9
   The multiplier in AB is subtracted from the PP in ST the number of times specified by the 10's complement of the digit value (10 minus the digit value). STAT H is set to supply a hot carry for the subtract sequence. STAT D is set to add a 1 to the next digit of the multiplicand (equivalent to adding the multiplicand 10 times).
4. SAL(4–7) = invalid digit
   The definition of an invalid digit is dependent on whether the digit to be processed is the first digit of the multiplicand; i.e., the digit immediately following the sign. If it is the first digit, then any value in the range of 10 through 15 is considered invalid and sets the interrupt trigger. After the first digit has been processed, a value of 10 is permissible in SAL(4–7), provided that it was formed by an original value of 9 to which a 1 has been added because STAT D was set. Under these conditions, the value of 10 does not set the interrupt trigger, no addition cycles are required, and a carry is propagated to the next digit by setting STAT D.

The multiplier-to-PP addition or subtraction is done one byte at a time in the serial adder, with the AB byte gated true +6 if adding and complement if subtracting. The ABC and STC are both initially set to the L2 value and are decremented by 1 each time a byte is processed. When the ABC count is stepped to 000, F(4–7) is examined to determine whether further additions or subtractions are necessary. If so, the STC and ABC are again set to the L2 value, F(4–7) is incremented if subtracting or decremented if adding, and the multiplier is again added to or subtracted from the PP. The micro-order which steps the digit in F(4–7) is executed after the digit has been examined to determine whether further add or subtract cycles are required. For this reason, when a branch on F(4–7) is being made, a value of 1 when adding or of 9 when subtracting indicates that the multiplicand digit is completely processed. The low-order digit of the PP in ST is the product digit developed.

Product Byte Store, PP Right-4 Shift to Drop Digit, Multiplicand Request

These three functions are accomplished in parallel fashion. After initiating the store operation, control is transferred to the shift-right-4 sequence. When the ST contents have been temporarily transferred, the store operation is resumed; the product byte is cross-gated, transferred to ST, and stored into main storage per the D-address. Thereafter, the microprogram requests the next multiplicand byte from main storage and simultaneously completes the right-4 shift.

As illustrated in Figure 3-19, the PP is shifted right-4 via the parallel adder. This shifting is done in several steps, with the LSWR being used as temporary storage for the operand. Upon completion of the right-4 shift, B(64–67) is normally inserted as the high-order S digit. B(64–67) was previously set to 0 if the value in F(4–7) was less than 5, or to 9 if F(4–7) was 5 or greater, for then the PP was in 10's complement form. An exception is made when STAT G is set, indicating an all-zero multiplier. In this case, B(64–67) is not inserted into high-order PP because it should always be zero.

Complete Multiplicand Byte Fetch

If there are more multiplicand digits to be processed (L1 $\neq$ L2), the contents of T are temporarily transferred to the LSWR and either the left or the right half of the operand is gated from the SDBO to T. The next byte of the multiplicand is then selected per the STC and transferred to F. (Note that if the left half word has been gated to T, the high-order STC bit is forced to 1, because, otherwise, the STC would select a byte from S.)

Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence.

The next digit of the multiplicand in F(4–7) is examined, STAT's D and H are set or reset as required, a 0 or 9 is placed into B(64–67), and the appropriate add or subtract loop is entered. After exit from the add or subtract loop, the low-order digit of the resulting PP is saved in F(4–7). A right-4 shift is then performed on the PP in ST so that the low-order digit is dropped. At the completion of the right-4 shift, the left-digit sequence is resumed. As explained previously, the contents of F are cross-gated and the next digit of the multiplicand is sampled from SAL(4–7).

Multiplicand Zero Test and Partial Product Store

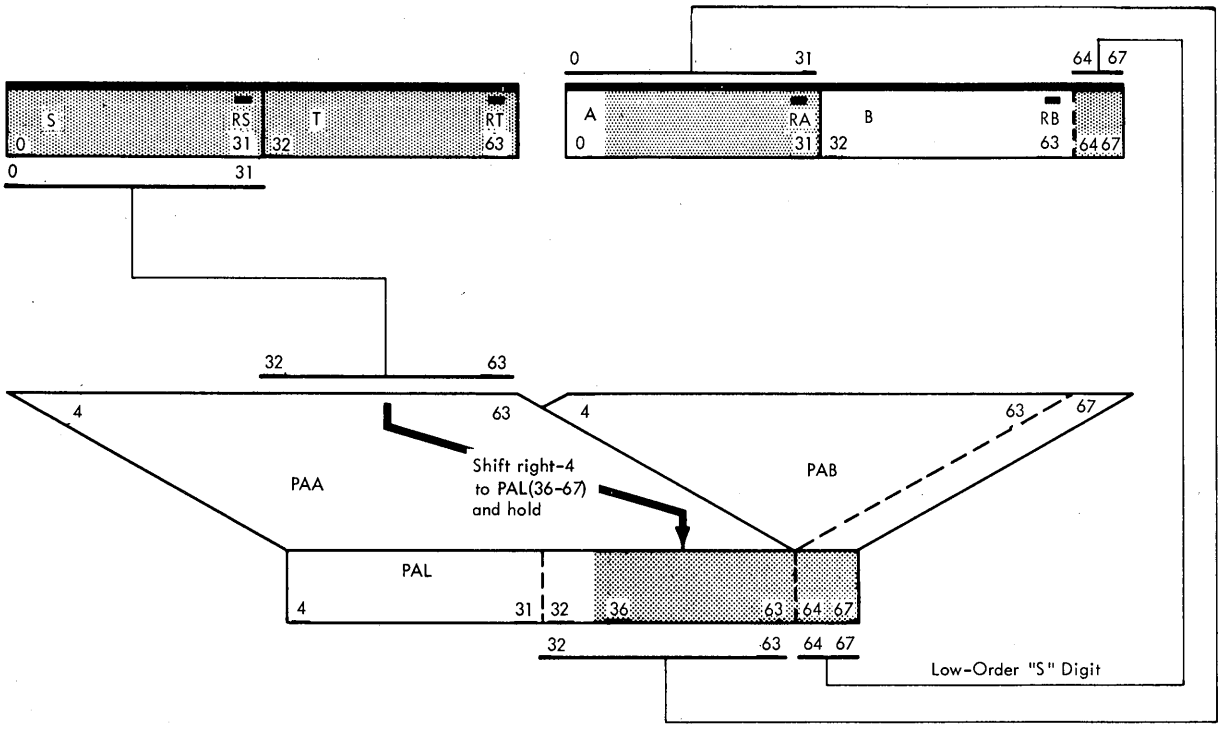
An entry to this routine is made from the left-digit sequence. By operand definition, the remaining high-order multiplicand bytes should all be zeros. The PP in ST is the high-order product and must be stored into the high-order portion of the initial multiplicand field. However, if STAT D is set at this time, the multiplier must be added to the PP once more. After this has been done, the contents of ST are transferred to AB and the high-order multiplicand bytes are fetched to ST (per the D-address). The STC is set per D(21–23) to designate the first high-order multiplicand byte to be tested for zero. The ABC is set per L2 to designate the first high-order PP byte in AB.

The selected multiplicand byte in ST is tested for zero; then the selected PP byte in AB is transferred via the serial adder to ST, and the corresponding mark trigger is set. ABC, STC, and L1 are decremented by 1 for each byte transferred. If a nonzero byte is detected in the high-order field of the multiplicand, the interrupt trigger is set and the instruction is ended.

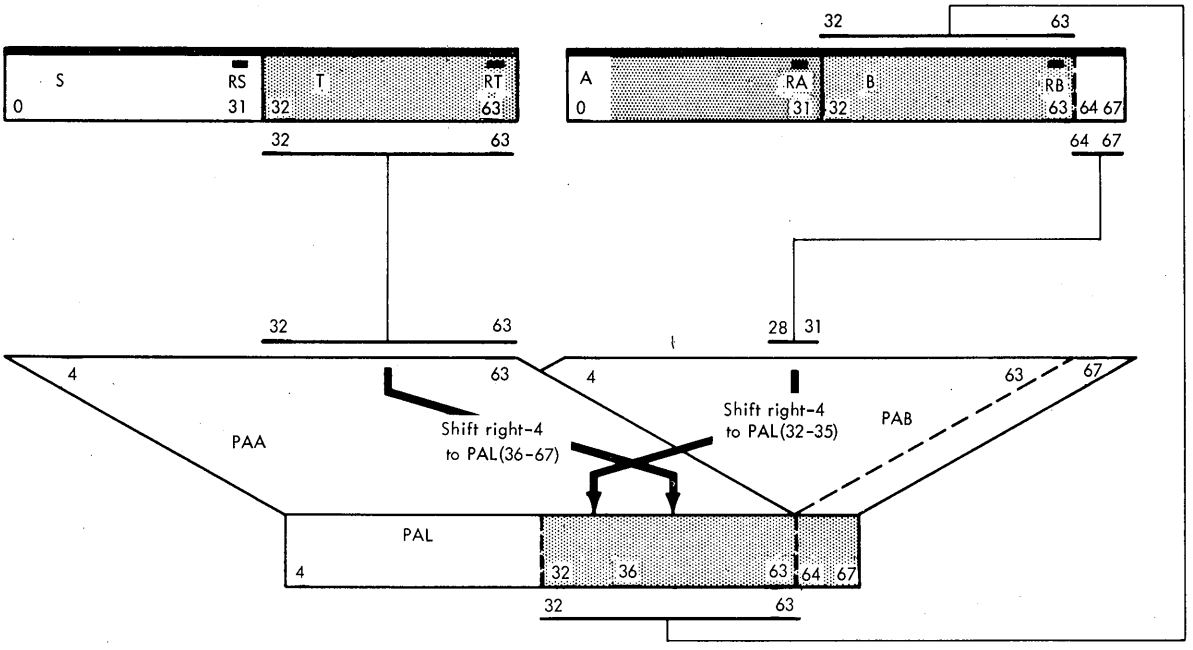
The AB-to-ST byte transfer is continued until L1 or STC is stepped to zero, at which time the ST contents are stored into main storage. If the STC has been stepped to zero (L1 $\neq$ 1111), the next high-order bytes of the multiplicand are fetched to ST and the sequence is resumed. If L1 has been stepped to zero (L1 = 1111), the instruction is ended.

DIVIDE, DP (FD)

- Divide 1st operand (in storage) by 2nd operand (in storage) and place result into 1st operand location (quotient is leftmost in 1st operand location; remainder, rightmost).



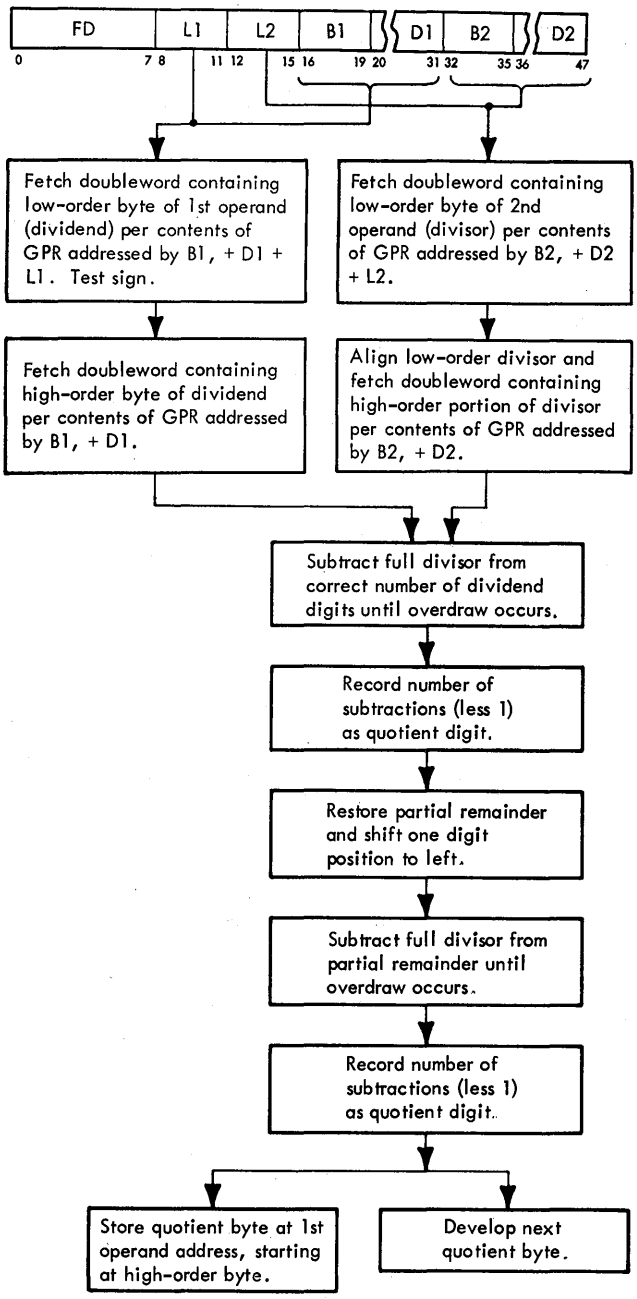
A. Step 1



B. Step 2

Figure 3-19. Data Flow for Right-4 Shift of ST to AB, Decimal Multiply

- SS format:



- Maximum dividend field (1st operand) is 16 bytes.
- Maximum divisor field (2nd operand) is 8 bytes.
- L2 specifies byte length for divisor and remainder.
- L2 > 7 or L2 $\geq$ L1 causes specification program interruption.
- Division accomplished by repetitive subtraction.
- Dividend field must initially contain sufficient number of high-order zeros to make possible storing of quotient and remainder.

The Decimal Divide instruction replaces the dividend (1st operand) with the quotient and the remainder. To be able to store the quotient and the remainder into the dividend field at all times, several restrictions are imposed on the initial size of the dividend and the divisor (Figure 3–20).

The maximum dividend field is 16 bytes long. It is eventually replaced by the quotient, which is stored leftmost in the field, and by the remainder, which is stored rightmost. The size of the remainder is equal to the initial divisor size and is therefore predefined by length code L2. Because the minimum remainder size is 1 byte (L2 = 0), the maximum quotient size is limited to 15 bytes. By definition, the size of the divisor (and remainder) cannot exceed 8 bytes. A divisor greater than 8 bytes, or in excess of the dividend, is recognized as a specification error; the instruction is suppressed and a specification program interruption occurs.

The operand signs are tested for validity before instruction execution, and, if either sign is invalid, a data program interruption is taken before the contents of main storage are altered.

To make sure that the quotient and remainder will fit into the destination field, the magnitudes of the dividend and the divisor are compared before entering the divide sequence. This comparison, called "divide check" or "trial subtraction," yields the number of quotient digits that will result if division is carried out. If the predicted quotient is larger than that allowed, the instruction is suppressed and a decimal divide program interruption occurs. For this reason, an overflow condition cannot exist upon execution of a Divide instruction.

The dividend, divisor, quotient, and remainder are all signed integers, right-aligned in their fields. The sign of the quotient is determined algebraically from the dividend and divisor signs. The sign of the remainder is the same as the sign of the dividend. These rules hold true even when the quotient or the remainder is zero.

The divide operation is executed in much the same manner as in manual arithmetic. First, the divisor is properly aligned with the high-order dividend; then, by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient digit is determined. The partial remainder resulting from the last successful subtraction is shifted one digit position to the left, and the next lower-order dividend digit is inserted at the low-order end of the partial remainder. To obtain the next quotient digit, the divisor is again subtracted from the partial remainder.

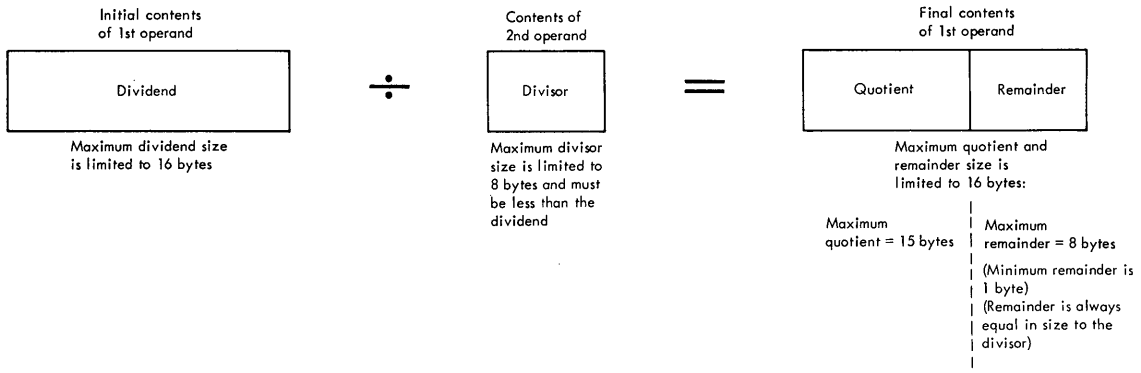


Figure 3-20. Operand Specifications for Decimal Divide

This sequence is repeated until all dividend digits have been processed. The remainder resulting from the final successful subtraction is given the sign of the dividend and stored into the low-order end of the dividend field.

Figure 3-21 illustrates a typical repetitive subtract sequence used to accomplish division. Initially, a sufficient number of high-order dividend digits must be selected to perform the first successful subtraction. Successful subtractions of the divisor from the dividend occur until the partial remainder is overdrawn. The divisor is then added back once to restore the correct partial remainder. At the same time, the quotient digit is decremented by 1 to compensate for the overdraw. As each dividend byte is processed, the length code of the dividend (L1) is reduced by 1 and compared with the length code of the divisor (L2). Because the size of the remainder is also defined by length code L2, the condition of L1 equal to L2 indicates that all the effective bytes of the dividend have been processed, and the remainder is to be stored into the rest of the destination field. Note that, to be able to fit the quotient and the remainder into the destination field, this field must initially contain high-order zeros. A data program interruption occurs if the dividend does not have at least one leading zero.

Following are general and detailed descriptions of the divide microprogram. The general description outlines the overall structure of the microprogram, enumerates its major functional steps and subroutines, and explains their relationship to the overall operation. The detailed description analyzes each sequence individually, making specific references to the register-to-register data transfer in the CE.

General Description

Upon entering the divide microprogram, the following actions have been performed by SS I-Fetch:

1. A D-request has been issued for the doubleword containing the low-order dividend byte.
2. The low-order dividend address has been placed into D.
3. The contents of the IC have been transferred to the LSWR, and the high-order divisor address has been placed into the IC.

An overall flowchart of the divide microprogram and the general data path used for its execution are shown in Sheet 1 of Diagram 5-306, FEMDM. The major subroutines and functional steps, shown in the figure, are explained in the following subparagraphs. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do not show the gates and data paths used in the CE, but are intended solely to convey how the divide algorithm is implemented. For purposes of illustration, a nine-byte dividend and a three-byte divisor are assumed in these diagrams.

General Initialization Sequence

This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence. GIS gates the low-order dividend from the SDBO to ST, increments the IC by L2 to address the low-order byte of the divisor, issues an IC request for the divisor, and gates the divisor to AB. The ABC is set to the low-order divisor byte in AB (Figure 3–22), and the STC is set per L2. STAT E is set if the dividend sign is invalid, and D is decremented by L1 to the address of the high-order byte of the dividend.

Specification Test

This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies eight bytes or less and is smaller than L1.

Incorrect Specification

Detection of an invalid specification forces a specification program interruption. The instruction address is restored to the LSWR, and the instruction is ended.

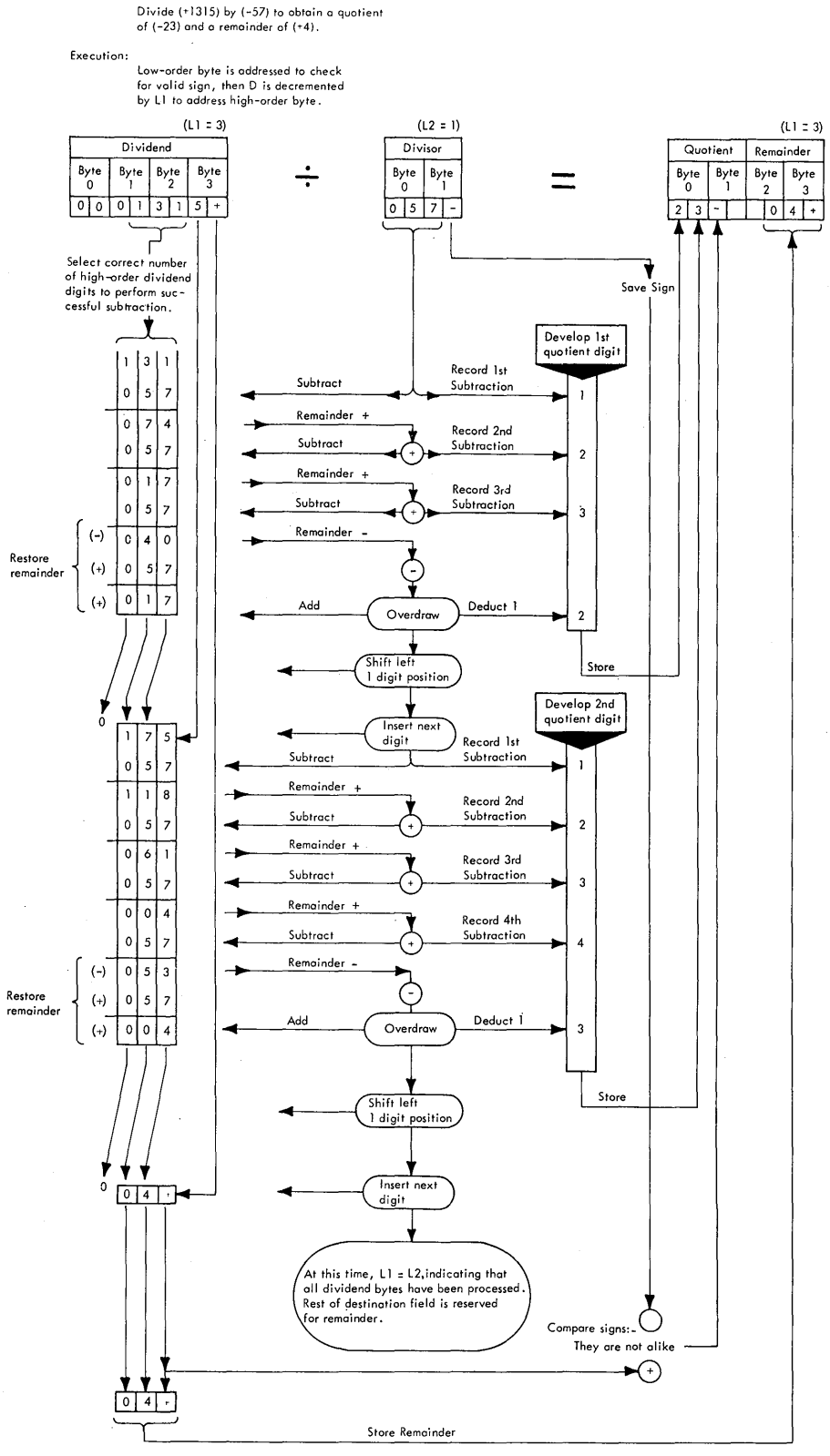


Figure 3-21. Example of a Typical Divide Sequence

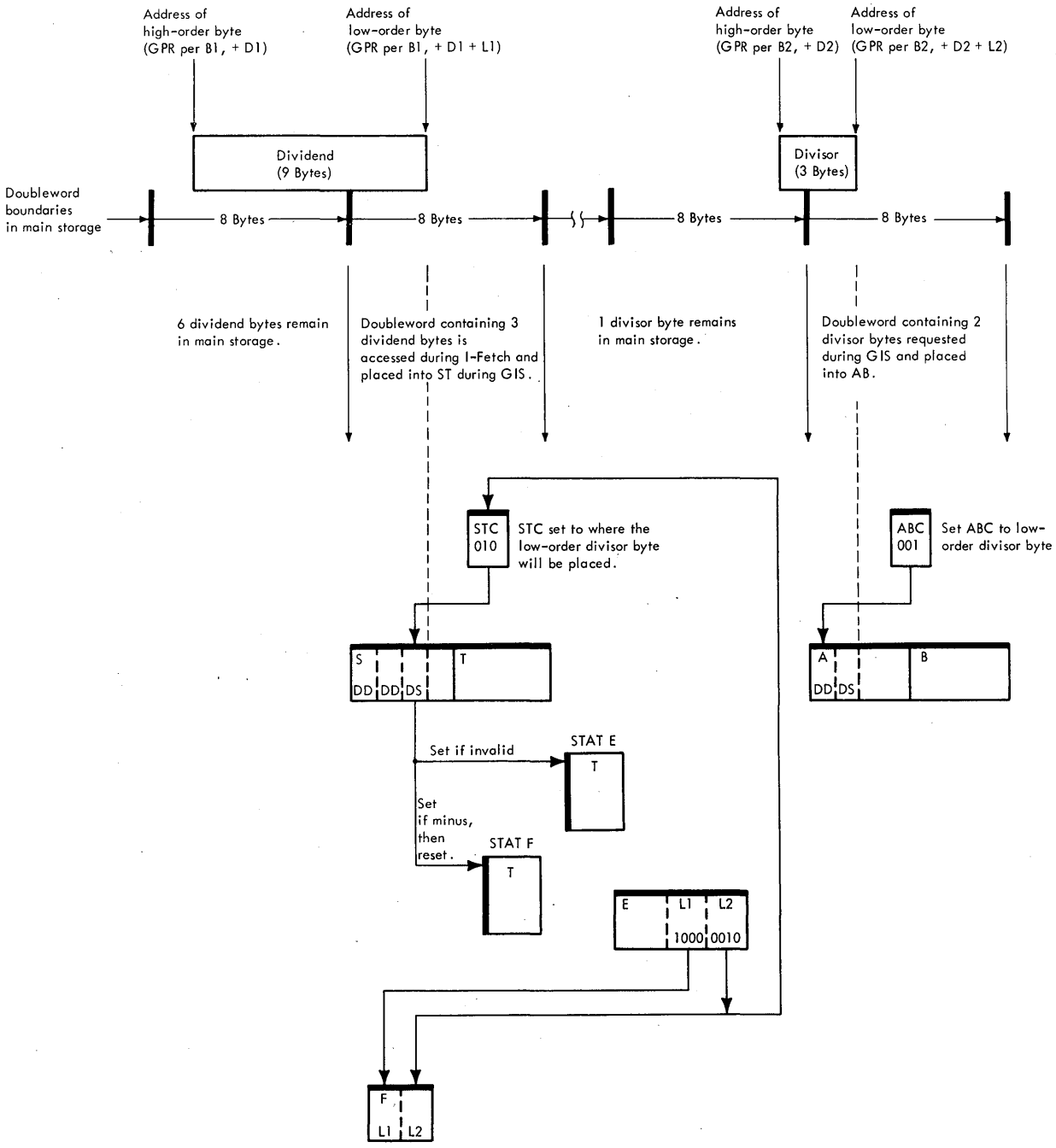


Figure 3-22. Data Handling During GIS of Decimal Divide

Divisor Left-Adjust Sequence

The divisor bytes are transferred from AB to ST in such a manner that the high-order divisor byte occupies the leftmost byte in ST. STAT C is set if the divisor sign is negative.

The left-adjust transfer is initiated by setting the STC per L2 (a function performed during the GIS). Because the maximum divisor length is limited to eight bytes, only three of the four bit positions in L2 are needed to effectively specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for one byte) to a maximum of 0111 (for eight bytes). Setting the STC per L2 automatically selects, according to the divisor size, the correct ST position for the low-order byte of the divisor; the number of bytes to the left of the selected ST position corresponds to the length field of the full divisor (Figure 3–23). The actual transfer is performed one byte at a time, through the serial adder, starting with the low-order divisor byte. ABC, STC, and L2 are decremented by 1 for each byte transferred. The divisor is completely transferred when L2 is decremented to zero. Because the first IC request (during I-Fetch) does not necessarily access the full divisor to AB, it may be necessary to fetch the balance of the divisor from main storage. (This fetch occurs if the ABC steps to zero before L2 steps to zero.)

After exit from the Divisor Left-Adjust sequence, the full divisor has been fetched and left-adjusted to ST.

Dividend Fetch and Left-Adjust Sequence

This sequence fetches a sufficient number of high-order dividend bytes to perform a trial subtraction of the divisor from the dividend. The full divisor is subtracted once from the high-order dividend. Because the maximum divisor size is eight bytes, eight high-order dividend bytes are required for trial subtraction. If the dividend is eight bytes or less, it is completely fetched during this sequence; if greater than

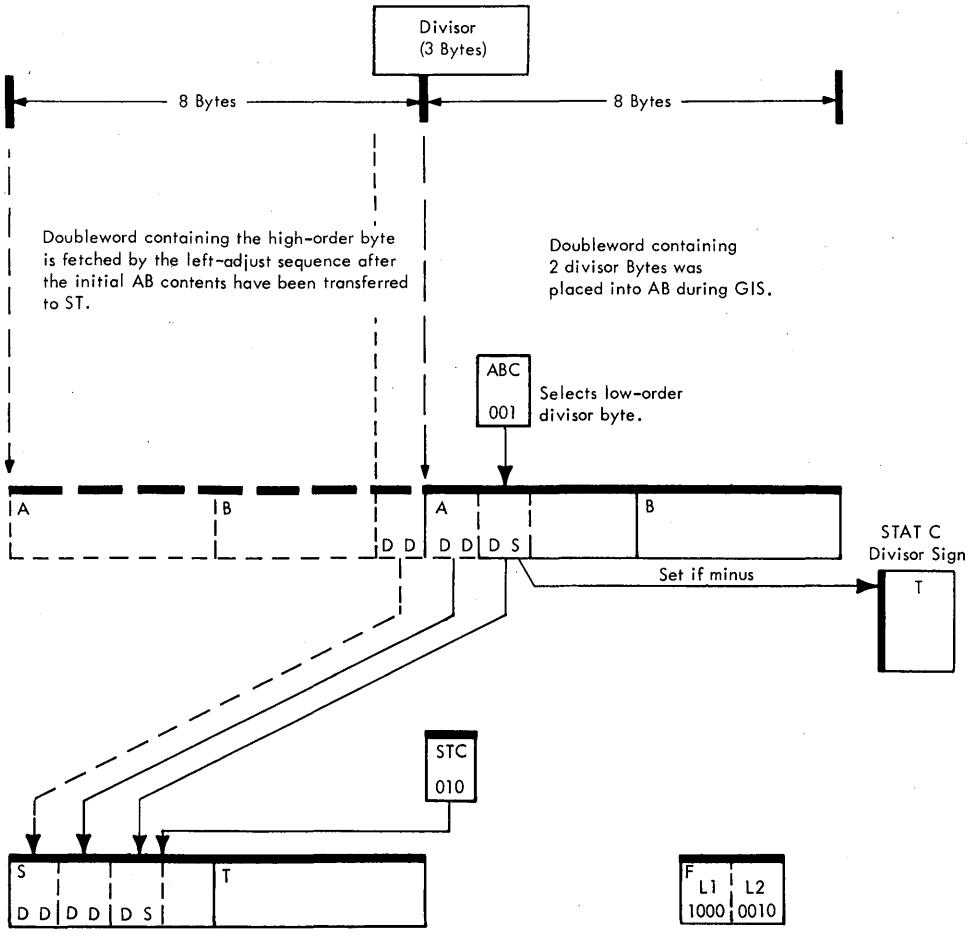


Figure 3-23. Data Handling During Divisor Left-Adjust Sequence

eight bytes, only the first eight high-order bytes are fetched. The dividend is fetched to AB and then transferred to ST in such a manner that the high-order byte occupies the leftmost byte in ST.

Upon entry into this sequence, ST is assumed to be completely occupied by the divisor. (If the divisor is four bytes or less, it is confined solely to S; if greater than four bytes, the divisor extends into T.) Because left-adjustment of the dividend requires the use of ST, the divisor must be transferred from ST; S is gated to the parallel adder and held in PAL, and T is stored into the LSWR (Figure 3–24).†

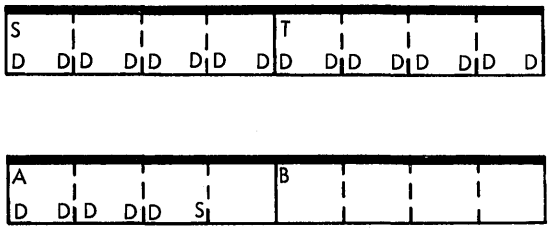
A request per the high-order dividend address is issued from D. Upon arrival of the dividend from main storage, the SDBO is gated to AB. The ABC is set per D(21–23) to select the high-order dividend byte. The left-adjust transfer is initiated by setting STC to 000, thus selecting the leftmost byte in ST. The dividend bytes are then transferred to ST, starting with the high-order byte. (The actual transfer is performed one byte at a time through the serial adder.) The ABC and STC are incremented, and L1 is decremented by 1 for each byte transferred. If L1 steps to zero before the ABC or STC steps to 7, the full dividend has been fetched and left-adjusted to ST. Because the first request does not necessarily access eight bytes of dividend to AB, it may be necessary to fetch additional dividend bytes from main storage. This fetch occurs if the ABC steps to 7 before the STC steps to 7 or L1 steps to zero.

Restore L1 and L2 to E

Left-adjustment of the divisor and dividend has decremented L2 and L1 to zero. The initial L2 and L1 counts, saved in F during GIS, are now restored to E(8–15). These counts are required by the subsequent divide sequence.

Assemble Divisor in AB and Dividend in ST

The divisor in PAL and LSWR is restored to AB. Upon completion of this function, both operands are left-aligned: the dividend is in ST, and the divisor is in AB.



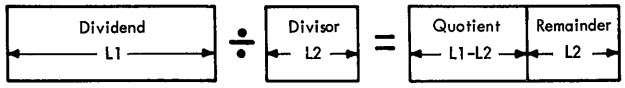
† The '→ HOLD' micro-order is issued on each cycle of the left-adjust sequence to hold the S contents in PAL.

Trial Subtraction

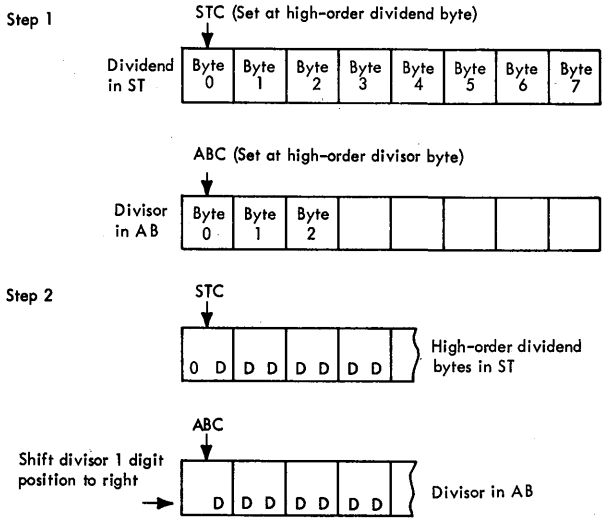
The divisor bytes in AB are subtracted from an equivalent number of high-order dividend bytes in ST. The remainder is then examined to establish whether the divide operation to follow will generate a result (quotient plus remainder) that will fit into the destination field. A negative remainder indicates that the destination field specified in the instruction is sufficiently large to accommodate the result. A positive remainder, however, indicates that the result cannot fit into the destination field, and a decimal divide program interruption occurs.

How prediction by trial subtraction is possible may be understood from the following considerations:

1. By definition, the dividend is at least one order higher than the divisor. The high-order digit position in the dividend is always zero.
2. The length code of the divisor (L2) is also the length code for the remainder. Consequently, the maximum number of quotient bytes that will fit into the destination field is equal to L1 minus L2. By operand definition, the difference of L1 minus L2 may range from a minimum of one byte to a maximum of eight bytes.



3. To perform the trial subtraction, the high-order divisor digit is aligned with the high-order digit of the dividend. This is performed in two steps: (1) the high-order divisor byte is aligned with the high-order byte of the dividend, and (2) because by definition the high-order digit position in the dividend is always zero, the divisor is shifted right one digit position to align the significant digits in both operands.



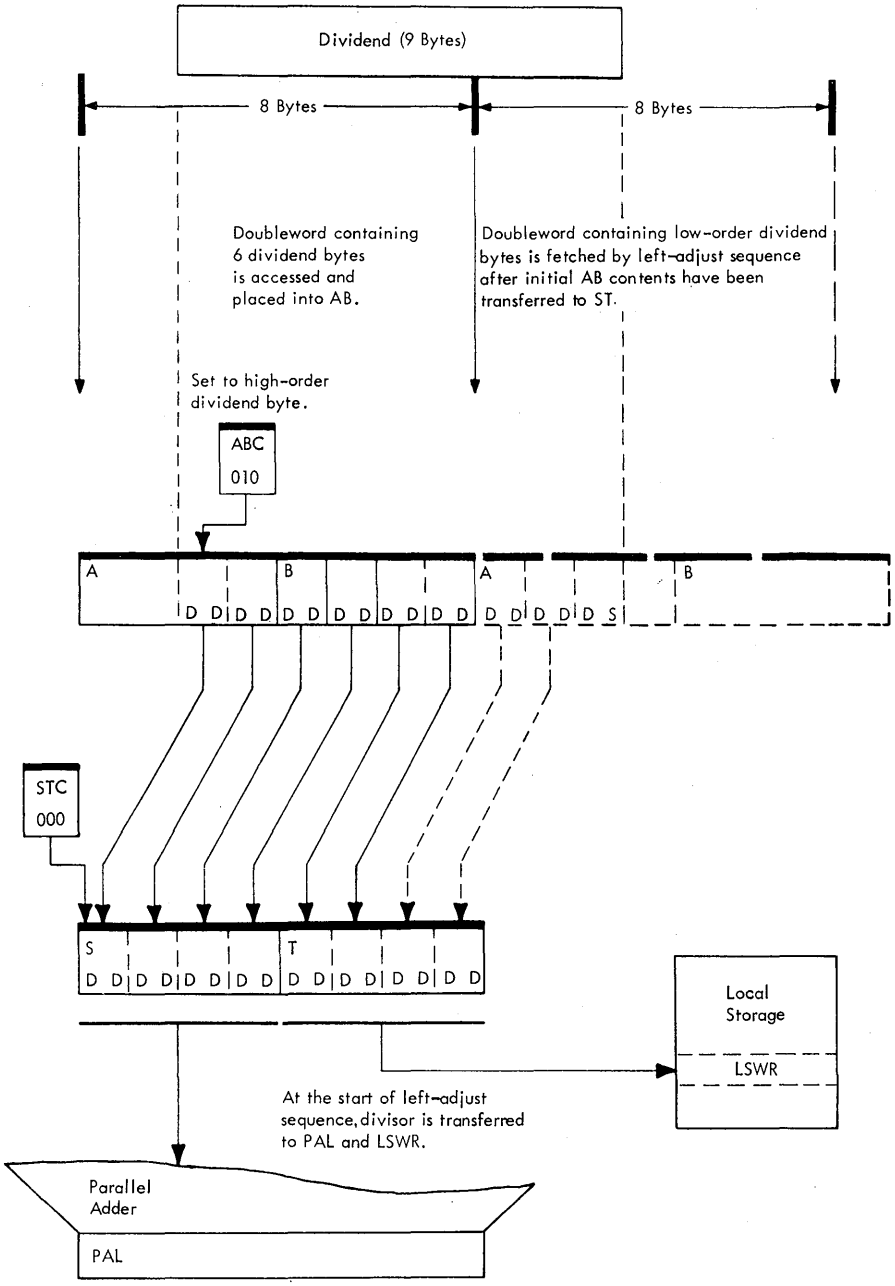
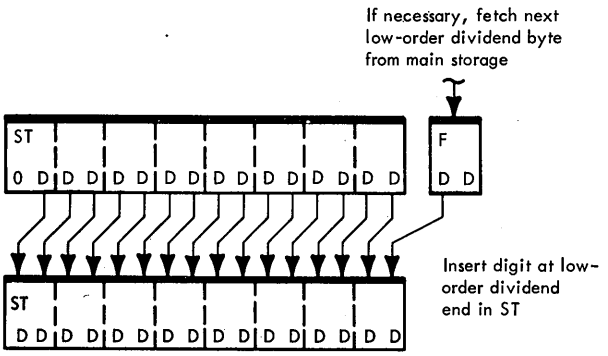


Figure 3-24. Data Handling During Dividend Fetch and Left-Adjust Sequence

4. Because the dividend is at least one order higher than the divisor, alignment of the high-order divisor digit with that of the dividend is equivalent to multiplying the divisor at least 10 times; if the dividend is one order higher, the divisor is multiplied 10 times; if two orders higher, 100 times; if three orders higher, 1000 times; and so on. Thus, during the trial subtraction, a quantity at least 10 times that of the divisor is subtracted from the dividend.
5. Because the maximum number of quotient digits allowed (L1 minus L2) corresponds to the difference between the orders of magnitude in the two operands, the result of the trial subtraction must always yield a negative remainder; otherwise, the number of quotient digits that would be generated would not fit into the destination field.

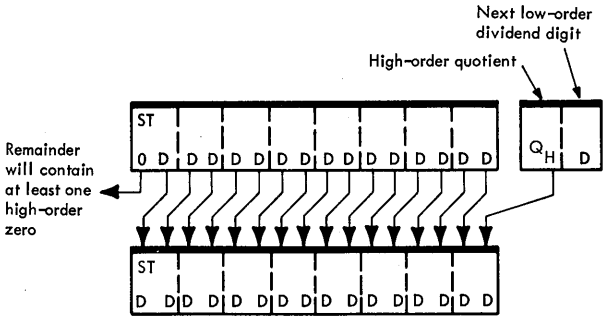
Shift Dividend One Digit to Left

The dividend is shifted one digit to the left to allow a successful subtraction of the divisor from the dividend. (To develop the quotient digit, the divisor must be repeatedly subtracted from the dividend until a negative remainder occurs.) Upon initiating the left-4 shift, a test is made to establish whether an additional low-order dividend digit is required for generation of the first quotient digit. If required, the next low-order dividend byte is fetched from main storage and placed into F. The digit is selected from F(0–3) and inserted at the low-order end of the dividend in ST.



Generate Quotient and Left-Digit Sequence

The divisor is repeatedly subtracted from the dividend until an overdraw occurs; i.e., a negative remainder is obtained. The number of successful subtractions is recorded and, after the last successful subtraction, becomes the high-order quotient digit. This digit is tested for validity and then inserted into F(0–3) by the left-digit sequence. This sequence also shifts the partial remainder (in ST) one digit to the left and inserts the next low-order dividend digit into the low-order end of ST.



Correct Low-Order Remainder Byte

In certain cases, the low-order remainder byte in ST must be corrected. The need for correction will become apparent when the Divide instruction is analyzed in detail. (See "Detailed Description" below.)

Generate Next Quotient Digit and Right Digit Sequence

After exit from the left-digit sequence, the operand length codes (L1 and L2) are compared to establish whether the last byte of the quotient is being processed. If L1 equals L2, the correct quotient sign is inserted into F(4–7). The last quotient byte (in F) is stored; then, the partial remainder (in ST) is stored into the low-order destination field as the final remainder.

If L1 does not equal L2, the next quotient digit is generated and placed into F(4–7) by the right-digit sequence. At the completion of this sequence, one complete byte of quotient is contained in F. This byte is stored into main storage, and the sequence for the left digit of the next quotient byte is started.

Detailed Description

- STAT A is set to indicate nonzero divisor.
- STAT C is set if divisor is negative.
- STAT D is set if dividend is less than eight bytes.
- STAT E is set if digit or sign is invalid.
- STAT F is set if dividend is negative.
- STAT G is first set if divisor is five bytes or greater. STAT G is then set again to enter left-digit sequence.
- STAT H is set to generate hot carry for subtract sequence.

Sheet 2 of Diagram 5-306 is a detailed flowchart of the divide microprogram. This figure is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the Divide instruction. The major subroutines and those areas in need of clarification are explained in the following subparagraphs.

General Initialization Sequence

This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

To test for an invalid dividend sign before the dividend is altered in main storage, the low-order dividend is accessed by a D request during SS I-Fetch. At the start of GIS, the doubleword containing the low-order byte of the dividend is gated to ST. The IC is incremented by L2 to address the low-order divisor byte, and the low-order divisor is accessed by an IC request. The contents of D are transferred to the STC, length codes L1 and L2 are transferred to F, the ABC is set to 7, and the IC is decremented by 8 to address the next doubleword of the divisor.

The dividend sign is tested, and STAT E is set if the sign is invalid. STAT F is set if the dividend sign is negative, but is immediately reset. D contains the address of the low-order byte of the dividend (contents of GPR addressed by B1, + D1 + L1). Because the divide operation begins at the high-order byte of the dividend, D is decremented by L1 to address the high-order byte of the dividend (contents of GPR addressed by B1, + D1). The STC is set per L2.

A test of L2 (contained in the STC) is performed to establish the byte size of the divisor. STAT G is set if the divisor is equal to or greater than five bytes. This function increases the execution speed when assembling the divisor in AB (see "Assemble Divisor in AB and Dividend in ST"); i.e., if the divisor is four bytes or smaller, the LSWR need not be restored to B.

Divisor Left-Adjust Sequence

1. The initial STC setting selects the rightmost ST byte that contains the low-order divisor byte.
2. STAT C is set if the divisor sign is negative.
3. If the ABC steps to zero before L2 steps to zero, the remaining low-order divisor bytes are fetched from main storage.
4. The divisor digits are checked for validity, and STAT E is set if an invalid digit is detected. STAT A is set to indicate a nonzero divisor. Division by zero results in a decimal divide program interruption during trial subtraction.
5. Upon fetching the full divisor, the divisor address is no longer needed, and the instruction address is restored to the IC.

Dividend Fetch and Left Adjust Sequence

1. The divisor is shifted one digit position to the right so that its high-order digit will be aligned with that of the dividend. (The dividend is not yet available.)
2. The low-order divisor word is transferred from T to the LSWR. The high-order divisor word is gated to the parallel adder and held in PAL by the '- → HOLD' micro-order.
3. The STC is set to zero to select the high-order ST byte (where the high-order dividend byte will be placed).
4. A test of the high-order L1 bit is performed to establish the byte size of the dividend. STAT D is set if the dividend is less than eight bytes. This function increased the execution speed upon exit from the trial subtraction. A branch per STAT D is made to determine whether the complete dividend has been fetched. (If STAT D is set, the full dividend has been fetched, because at least eight dividend bytes are fetched to perform the trial subtraction.)
5. If the ABC steps to zero before L1 or STC steps to zero, the D-address is incremented by 8 and a fetch of the next doubleword of the dividend is made. The destination address for the subsequent quotient bytes is then restored by subtracting 8 from D.

Assemble Divisor in AB and Dividend in ST

This function is performed in parallel with the L1 and L2 restoration sequence. The high-order divisor word is transferred from PAL to A, after which restoration of the L1 and L2 counts is started. During the restoration sequence, STAT G is tested to establish whether the full divisor has been assembled in AB. If STAT G is not set, the divisor is four bytes or less. Therefore, the full divisor was contained in PAL and has been placed into AB. In this case, the restoration sequence is completed and an exit is made to the trial subtraction routine.

If STAT G is set, the low-order portion of the divisor is contained in the LSWR and must be transferred to B before entering the trial subtraction. The LSWR contents must be transferred to B via ST, which contains the left-aligned dividend. Several execution cycles are used to transfer the LSWR contents to B without destroying the ST contents.

Trial Subtraction

The divisor is subtracted from the dividend one byte at a time. After the last subtract cycle, a branch is made on a carry condition from SAL(0). Presence of a carry indicates that the remainder is positive, and the instruction is ended. Absence of a carry indicates a negative remainder, and that the result of the divide operation will fit into the destination field.

Dividend (or Partial Remainder) Left-4 Shift

The dividend is shifted left one digit position to perform the first successful subtraction of the divisor from the dividend. Upon initiating the left-4 shift, a test (per STAT D) is made to establish whether an additional dividend digit must be inserted into the low-order end of ST. If STAT D is set (see "Dividend Fetch and Left-Adjust Sequence," step 4), all dividend bytes have been fetched from main storage and the left-4 shift is completed. If STAT D is not set, the following actions take place:

1. The D-address is incremented by 8, and the next doubleword of the dividend is requested from main storage.
2. The contents of T are temporarily transferred to the LSWR. (Upon arrival of the dividend doubleword from main storage, T is loaded with the dividend word containing the next digit to be inserted.)
3. The STC is set per D(21–23) to select the correct dividend byte in the requested doubleword.
4. The left-4 shift of the dividend is completed. The high-order dividend word is in S, and the low-order word is in the LSWR.
5. A branch per D(21) is made to establish which word in the SDBO contains the next dividend byte. The correct word is then gated from SDBO to T. [Note that, if the left SDBO word is gated to T, STC(0) is forced to 1 to select the correct byte in T.]
6. The selected dividend byte is transferred from T to F. The shifted low-order dividend word is then restored from the LSWR to T.
7. The destination address is restored by subtracting 8 from D.
8. The high-order L1 bit is tested to establish the byte size of the dividend, and STAT D is set if the dividend is less than eight bytes. This function increases the execution speed upon exit from the right-digit sequence.

Generate Quotient Sequence

1. The ABC and STC are set per L2 to select the low-order operand bytes. STAT H is set to provide a hot carry to the serial adder.
2. The selected AB byte is subtracted from the selected ST byte via the serial adder. The result is gated back to the selected ST byte, with the carry being saved in STAT H. Any invalid digit detected in the serial adder sets STAT E.
3. The ABC and STC are decremented as each byte is processed. When the ABC is stepped to zero, a 1 is added to F(4–7) and the ABC and STC are again set per L2.
4. If a serial adder carry results upon processing the high-order byte, the partial remainder in ST is positive and the divisor is again subtracted from the dividend. F(4–7) is incremented by 1 each time a complete subtraction is made.
5. If there is no carry upon processing the high-order byte, an exit is made to the appropriate left- or right-digit sequence, as determined per STAT G.
6. Note that, before starting each subtract sequence, the partial remainder resulting from the previous subtraction is saved in the LSWR and PAL. The saving is done because, upon exit on a no-carry condition, an overdraw has occurred and the remainder in ST cannot be used for computation of the next quotient digit. Instead, the partial remainder resulting from the last successful subtraction is used for subsequent computation.

Left-Digit Sequence

1. The quotient digit in F(4–7) is the left digit of a quotient byte. This digit is reduced one count, to compensate for the overdraw, and then cross-gated via the serial adder to F(0–3).
2. The partial remainder resulting from the last successful subtraction and saved in the LSWR and PAL is shifted one digit to the left and restored to ST. The next low-order dividend digit, in B(64–67), is inserted into the low-order end of ST.
3. A test on STC equal to or greater than 4 is made to establish whether the low-order byte of the partial remainder in the LSWR has been overdrawn. In the generate quotient sequence, the contents of T are stored into the LSWR at the same time that the first subtract cycle is performed. Thus, if the partial remainder extends into T (which occurs if the STC is 4 or greater), the low-order divisor byte is subtracted from the low-order partial remainder byte once too often. In such cases, the low-order byte of the partial remainder is corrected by adding it to the low-order divisor byte. After performing the correction, the left-digit sequence is re-entered.
4. If L1 equals L2, the quotient sign byte is processed. Otherwise, the quotient digit generation routine is resumed to develop the next digit.

Correct Low-Order Remainder Byte

This routine is entered from the left- or right-digit sequence if the low-order divisor byte has been subtracted once too often from the low-order byte of the partial remainder. Correction is performed as follows:

1. STAT H is reset to initiate a true add cycle.
2. The low-order partial remainder word is placed into T. The STC is set per L2 to select the low-order byte in T.
3. The low-order divisor byte (per the ABC) is added once to the low-order partial remainder byte (per the STC), and the result is gated to T per the STC.
4. The left- or right-digit sequence is re-entered, as applicable.

Right-Digit Sequence

This sequence is entered when two quotient digits have been generated and placed into F. The following actions are performed:

1. The STC is set per D(21–23), and F is transferred to the selected ST byte. The corresponding mark trigger is set per the STC.
2. A storage request is issued to store the quotient byte per the D address.
3. A left-4 shift of the partial remainder (in PAL and LSWR) is initiated.
4. If STAT D is set, indicating that a dividend byte fetch is not required, the left-4 shift is completed and the partial remainder is restored to ST.
5. If STAT D is not set, the dividend byte fetch sequence is entered.
6. D is decremented by 1 to obtain the destination address for the next quotient byte.
7. F is cleared and STAT G is set to enter the left-digit sequence, after the first quotient digit is generated.
8. If STAT E is set, the 'invalid data interrupt' trigger is set and the instruction is ended.
9. If STAT E is not set, the generate-quotient sequence is entered.

Process Quotient Sign Byte

This routine is entered from the left-digit sequence when L1 equals L2. At this time, all dividend digits have been processed: the low-order quotient digit is in F(0–3), the byte selected by the STC is the dividend sign byte, and the remaining high-order contents of ST are the final remainder. The following actions take place:

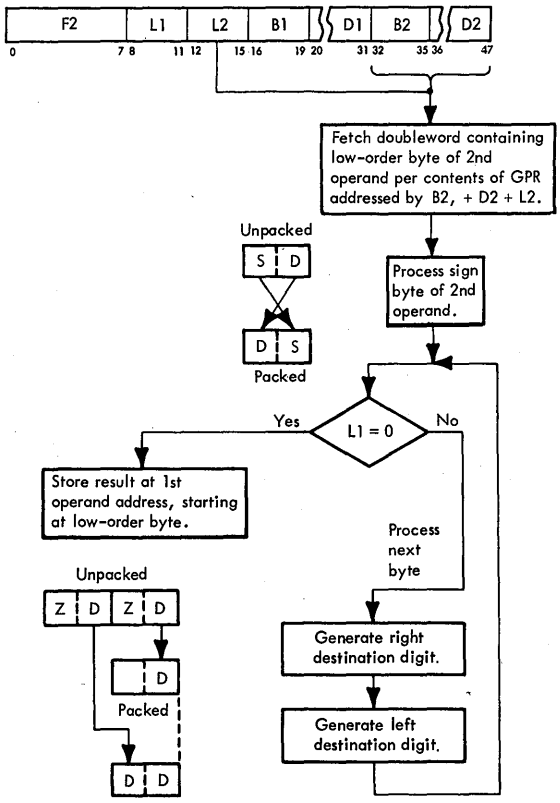
1. The STC and ABC are set per the L2 count. STAT F is set if bits 4–7 of the selected ST byte indicate a negative sign. STAT E is set if the sign is invalid; however, if an invalid sign existed, STAT E would have been set and a data program interruption would have occurred earlier.
2. The correct negative or positive sign is put into F(4–7) as determined by a comparison of STAT's C and F.
3. The ST contents are transferred to AB via the parallel adder.
4. The STC is set per D(21–23), and F is gated to the selected ST byte. The corresponding mark trigger is set, and the selected ST byte is stored into main storage.

Store Remainder Routine

1. The byte selected by the ABC, which is the low-order remainder byte, is saved in F. If necessary, the remainder sign is corrected in the serial adder before gating to F.
2. The remainder is transferred from AB to ST one byte at a time. As each byte is transferred, the corresponding mark trigger is set, the ABC and STC are incremented by 1, and L2 is decremented by 1.
3. When the STC steps to 7, ST contents are stored per the D-address. D is then incremented by 8, and the byte transfer is resumed.
4. When L2 steps to 0, the STC is decremented by 1, and the remainder sign byte is gated from F to ST. The contents of ST are then stored into the low-order destination field, and the instruction is ended.
5. If STAT E is set, an exit is made to the program interruption microprogram.

PACK, PACK (F2)

- Convert format of 2nd operand (in storage) from zoned to packed and place result into 1st operand location (in storage).
- SS format:



- Separate microprogram is used during word overlap.

The Pack instruction assumes source data in the unpacked format. The low-order source byte consists of a sign (bits 0–3) and a digit (bits 4–7). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes consist of a zone (bits 0–3) and a digit (bits 4–7). Only the digits are gated to the destination field, with two bytes of source being processed for each byte of destination.

The sign and digits of the second operand are moved unchanged to the first operand field and are not checked for valid codes. A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for word overlap is performed in the GIS of the instruction and also each time that a new doubleword of source is fetched from main storage.

The GIS microprogram for the Pack instruction is shown in Diagram 5-307, FEMDM. This microprogram gates the low-order first operand from the SDBO to ST, increments the IC by L2 to address the low-order byte of the second operand, gates the low-order second operand from the SDBO to AB, and performs the word-overlap test.

The word-overlap test is performed in two steps. First, the doubleword addresses for the destination and source are compared by subtracting D from the IC. The difference is then shifted four bit positions to the right and gated to PAL, and PAL(40–64) is sensed for an all-zero result to detect a possible word overlap. [The right-4 shift is made to avoid comparison of byte addresses within the doubleword; i.e., the difference for the byte addresses is shifted tc PAL(65–67), which is not sensed by the branch.] If the addresses for the doublewords of source and destination are different, no word-overlap condition exists. Thus, if PAL(40–64) is not zero, a branch is made to the appropriate not-word-overlap execution sequence of the instruction.

If PAL(40–63) equals zero, indicating that the same doubleword address has been specified for the source and destination, a second test must be made to verify whether special data handling is required. The contents of D are again subtracted from the IC, but this time a right-4 shift on the difference is not performed and the byte addresses within the same doubleword are compared. If PAL(40–63) equals zero, an identical address has been specified for both source and destination. Because this case of word overlap does not require special data handling, a branch is made to the not-word-overlap microprogram. If, however, PAL(40–63) is not zero, the source and destination bytes are skewed; special data handling is required in the execution phase and, accordingly, a branch is made to the appropriate program.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Process sign byte and test for exit conditions.
  2. If no exit conditions, process right destination digit.
  3. Process left destination digit and test for exit conditions.

A flowchart of the execution of the Pack instruction without word overlap is shown in Diagram 5-308, FEMDM. The major functional steps in the microprogram are described in the following subparagraphs.

Process Sign Byte

1. The selected AB byte is gated via the serial adder cross-gates to the selected ST byte. The mark trigger selected by the STC is set.
2. ABC, STC, L1, and L2 are decremented by 1.
3. An exit is made to the appropriate routine if one or more of the counters (ABC, STC, L1, L2) was equal to zero before being stepped.
4. If no exit is made, the next source byte is processed to obtain the right destination digit.

Generate Right Destination Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7); no data is gated to SAA(0–3). The serial adder output is gated from SAL(0–7) to the selected ST byte.
2. The ABC and L2 are decremented by one count.
3. If L2 equals zero before stepping, the remaining source bytes are extended with high-order zeros. (See "Extension of Source Bytes with High-Order Zeros.")
4. If the ABC equals zero before stepping, an exit is made to the source fetch routine. STAT G is set to cause a return to the Generate Left Destination Digit routine after the source fetch.

Generate Left Destination Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(0–3). Bits 4–7 of the selected ST byte are gated to SAB(4–7). The serial adder output is gated back to the selected ST byte, and the mark trigger selected by the STC is set.
2. ABC, STC, L1, and L2 are decremented by 1. If none of these counters equalled zero before stepping, the right digit for the next destination byte is generated. (Generate Right Destination Digit sequence is entered.)

Exit Conditions

An exit is made from the sign byte routine or from the left-digit routine when one or more of the following conditions are detected by the 'DECIMAL' (functional branch) micro-order:

1. L1 or STC = 0.
2. L2 = 0.
3. ABC = 0.

When the exit is on L1 or STC equals zero, a second test on L1-equal-all-1's is required to determine whether an end-op condition exists.

Extension of Source Bytes with High-Order Zeros

This routine is entered when L2 has stepped to zero before L1 has stepped to zero.

The serial adder output (zeros) is gated to the selected ST byte with the selected mark trigger being set. L1 and STC are decremented as each byte is processed. When L1 equals zero, the contents of ST are stored per the D-address

and the common end-op routine is started. When the STC equals zero, the contents of ST are stored, and D is decremented by 8. STAT H is set to cause a return to this routine after storing the contents of ST.

Source Fetch Routine

This routine is shared with the Move with Offset instruction. STAT D is set to cause a return to the pack microprogram.

The second operand is requested from main storage, and the IC is decremented by 8. A word-overlap test is performed. If no word-overlap condition exists, the next doubleword of the second operand is gated from the SDBO to AB. Processing of the left or right destination digit is resumed as determined by STAT G.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Process sign byte. Update AB and test for exit conditions.
  2. If no exit conditions, process right destination digit.
  3. Process left destination digit, update AB, and test for exit conditions.

A flowchart of the Pack instruction execution under word-overlap conditions is shown in Diagram 5-309, FEMDM. This microprogram is entered when a word-overlap condition is detected in the GIS or during a source fetch. The major functional steps in the microprogram are described in the following subparagraphs:

Process Sign Byte

The sign byte of the second operand in AB is processed in the same manner as in the not-word-overlap microprogram.

Update AB from ST

The data in AB is updated by transferring the contents of S to A or the contents of T to B, depending on the STC setting. ABC, STC, L1, and L2 are decremented by 1, and the mark trigger selected by the STC is set.

If any counter equalled zero before decrementing, an exit is made to the proper store, fetch, or extend-with-zeros routine as explained for the not-word-overlap sequence. If no exit conditions exist, processing of the right destination digit is started.

Generate Right Destination Digit

This routine is the same as in the not-word-overlap sequence.

Generate Left Destination Digit

This routine is the same as in the not-word-overlap sequence and is always followed by the update routine.

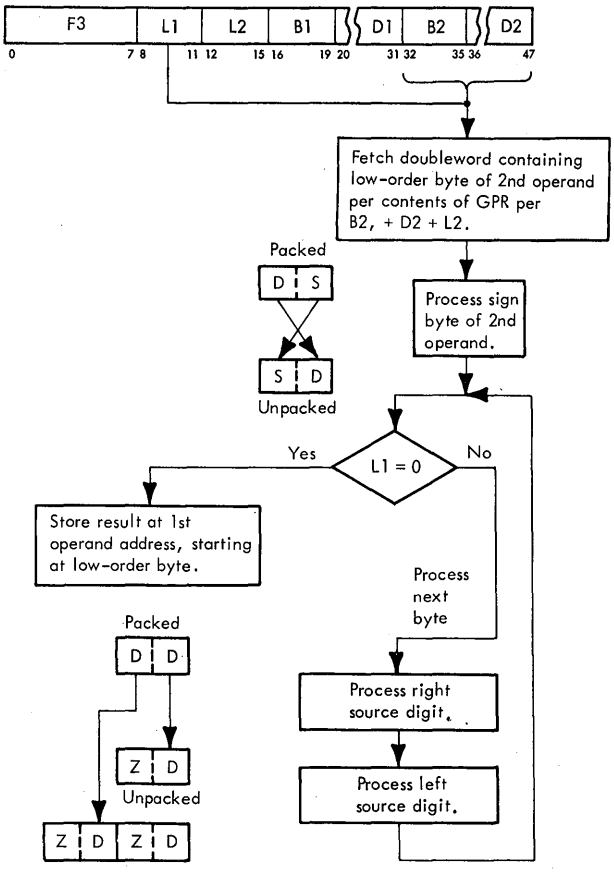
Source Fetch Routine

The next doubleword of source is requested from main storage, after which the IC is decremented by 8. Upon detection of a word-overlap condition, however, this doubleword is not used, because AB must be updated from ST. If, upon entering the source fetch routine, only the right destination digit has been placed into the selected ST byte, this byte is not transferred to AB. Instead, the following action takes place:

1. The portion of ST that has been processed (as determined by the mark triggers) is stored into the destination field, refetched from storage, and gated to both AB and ST.
2. If STAT G is set, indicating that only the right digit of the selected ST byte has been processed, the selected ST byte is transferred to F before the SDBO is gated to ST. After the SDBO is gated to ST, F is reinserted into the selected ST byte, and processing of the left digit is started.
3. If STAT G is not set, indicating that a complete ST byte has been processed, it is not necessary to save the selected ST byte. Processing of the right digit is started immediately.

UNPACK, UNPK (F3)

- Convert format of 2nd operand (in storage) from packed to zoned and place result into 1st operand location (in storage).
- SS format:



- Separate microprogram is used during word overlap.
- Word-overlap test is performed during GIS and in destination store and source fetch routines.

The Unpack instruction assumes data in the packed format. The low-order source byte consists of a sign (bits 4–7) and a digit (bits 0–3). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes contain a pair of binary-coded-decimal digits. Each digit is transferred to the low-order portion (bits 4–7) of the corresponding destination byte, and a zone character is inserted into the high-order portion byte (bits 0–3). During this transfer, the digits are not tested for validity.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for a word-overlap condition is performed in the GIS of the instruction and also each time that a doubleword of data is fetched from or stored into main storage.

The Unpack instruction generates two bytes of destination for each byte of source. Therefore, the condition when the destination bytes are processed "ahead" of the source always exists if the operand fields overlap. When the same doubleword address is specified, special data handling is required regardless of how the operand bytes are arranged in this doubleword. Special handling is necessary each time that source data is fetched from main storage; also, upon storing unpacked data into the destination field, a word-overlap test must be made to determine whether the source data in the CE must be updated from storage.

The GIS microprogram for the Unpack instruction is shown in Diagram 5-307. When the first overlap indication occurs, the byte addresses are not checked. Instead, a branch is forced into the word-overlap sequence by supplying a hot carry to PAA(60), so that a test of PAL(40–63) always yields a nonzero result.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Process sign byte and test for exit conditions.
  2. If no exit conditions, process right source digit.
  3. Process left source digit, and test exit conditions.

A flowchart of Unpack instruction execution without word overlap is shown in Diagram 5-310, FEMDM. The major functional steps in the microprogram are described in the following paragraphs.

Process Sign Byte

The sign byte, selected by the ABC, is gated via the serial adder cross-gates to the selected ST byte, and the corresponding mark trigger is set. ABC, STC, L1, and L2 are decremented by 1, and an exit is made if any counter equalled zero before stepping.

Process Right Source Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7). The approved zone character is inserted into SAA(0–3). The serial adder output is gated to the selected ST byte, and the selected mark trigger is set.
2. L1 and STC are decremented by 1.
3. If L1 equalled zero before stepping, the contents of ST are stored and the common end-op sequence is started.
4. If STC equalled zero before stepping (and L1 was not zero), the destination store routine is started. STAT G is set to record an exit from the right digit routine.

Process Left Source Digit

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7), and the zone character is inserted into SAA(0–3).
2. The adder output is gated to the selected ST byte, and the selected mark trigger is set.
3. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero before stepping. If no exit condition exists, the right source digit in the next source byte is processed.

Exit Conditions

An exit is made from the byte processing routine whenever it is detected that L1, L2, ABC, or STC is equal to zero. Although a separate exit is provided for each possible combination of these conditions, they may be considered to be examined in the following order of priority:

1. L1 = 0
   The contents of ST are stored per the D-address, and the common end-op routine is started.
2. L2 = 0
   AB is cleared, the ABC is set per L2 (which is 7), and STAT H is set to record the end of the source field. If the STC was also zero, the destination store routine is started. If the STC was not zero, the high-order zeros routine is entered per STAT H.
3. STC = 0
   The destination store routine is started. If the ABC was also zero, STAT D is set to cause a source fetch after the destination store.
4. ABC = 0.
   The source fetch routine is started.

Extension of Source Bytes with High-Order Zeros

AB is cleared, and bits 4–7 of the selected AB byte (zeros) are gated to SAA(4–7); the approved zone character is inserted into SAA(0–3). The adder output is gated to the selected ST byte, and the corresponding mark trigger is set. L1 and STC are decremented by 1 for each byte that is

processed. An exit is made to the destination store routine when the STC steps to zero, and to end-op when L1 steps to zero.

Source Fetch Routine

A request is made per the IC address, after which the IC is decremented by 8. A word-overlap test is made. If there is no word-overlap condition, the next source word is gated to AB, and the right digit of the next source byte is processed.

Destination Store Routine

1. The contents of ST are stored into the destination field per the D-address, and D is decremented by 8.
2. An exit is made to the source fetch routine if STAT D is set.
3. An exit is made to the high-order zeros routine if STAT H is set.
4. If neither STAT D nor STAT H is set, a word-overlap test is made by comparing the IC and D addresses. If no word overlap exists, the left or right digit is processed as determined by STAT G.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Process sign byte. Update AB and test for exit conditions.
  2. If no exit conditions, process right source digit.
  3. Process left source digit, and test for exit conditions.
- Word-overlap test is performed during source fetch and destination store routines.

A flowchart of Unpack instruction execution under word-overlap conditions is shown in Diagram 5-311, FEMDM. The steps in which this microprogram differs from that for not-word-overlap are explained in the following paragraphs.

Process Sign Byte

This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

Update AB from ST

If the STC is less than 4, the contents of S are transferred to A; the contents of T are always transferred to B. The mark trigger selected by the STC is set. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero before their being stepped. If no exit conditions exist, the right digit in the next source byte is processed.

Process Right Source Digit

The right source digit is processed in the same manner as for not-word-overlap. If upon processing the right digit an exit is made on STC equal zero, and ABC is not zero, the contents of S are transferred to A. In this manner, the source is correctly updated before storing the contents of ST.

Process Left Source Digit

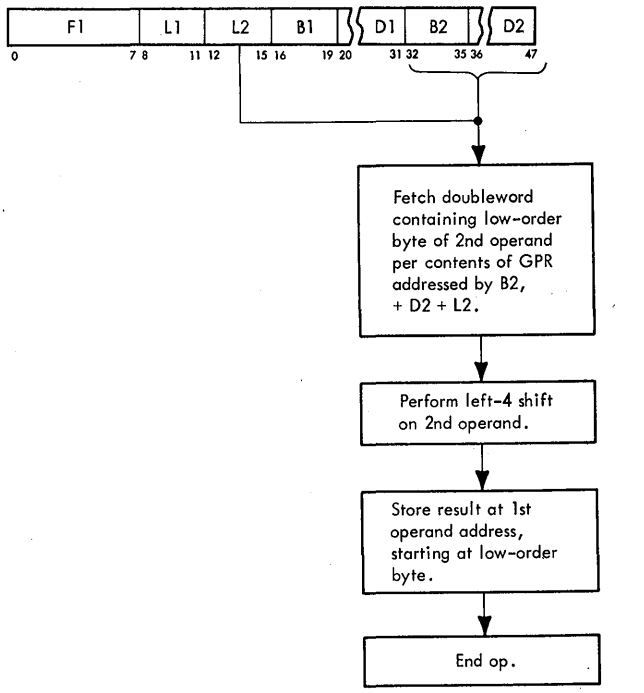
This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

Source Fetch Routine

1. The source is requested per the IC address, after which the IC is decremented by 8.
2. The contents of D are subtracted from the IC to prepare for the word-overlap test; also, a test on STC equals 7 is made to establish how the source is to be updated in case of an overlap condition.
3. The condition when STC equals 7 indicates that the STC was zero before entering the source fetch routine. In this case, the destination has been stored into main storage. Thus, to update the source, the doubleword at the SDBO is gated to AB and ST, and processing of the left source digit is started.
4. If the STC is not 7, AB must be updated from ST. After transfer of the contents of ST to AB, processing of the right source digit is started.

MOVE WITH OFFSET, MVO (F1)

- Store 2nd operand (in storage) to left of and adjacent to low-order 4 bits of 1st operand (in storage).
- SS format:



- Separate microprogram is used during word overlap.

The MVO instruction performs a left-4 shift on the second operand and transfers the result to the first operand location. Thus, the four low-order bits of the first operand are preserved as the lowest-order character of the second operand. During execution of the instruction, the operand signs and digits are not tested for valid codes.

No decimal shift instruction is provided, because the equivalent of a shift can be obtained by programming. Programs for right or left shift, and for an even or an odd shift amount, are written with Move with Offset instruction and the logical move instructions described in Section 5 of this Chapter.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for word overlap is performed in the GIS of the instruction, and also each time that a new doubleword of source is fetched from main storage.

The GIS for the Move with Offset instruction is shown in Diagram 5-307. This microprogram is identical with the GIS microprogram of the Pack instruction.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Transfer bits 4–7 of selected AB byte to bits 0–3 of selected ST byte. Decrement counters.
  2. Transfer bits 0–3 of selected AB byte to bits 4–7 of selected ST byte. Repeat first step.
  3. Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A flowchart of the execution of the Move with Offset instruction when no word-overlap condition exists is shown in Diagram 5-312, FEMDM. Basically, this microprogram specifies a 2-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

Cycle 1

1. Bits 4–7 of the selected AB byte are gated to SAA(0–3).
2. Bits 4–7 of the selected ST byte are gated to SAB(4–7).
3. The serial adder output is gated back to the selected ST byte, and the corresponding mark trigger is set.
4. L1 and STC are decremented by 1. An exit is made to the destination store routine if L1 or STC equalled zero before stepping.

Cycle 2

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7). No data is gated to serial adder bits 0–3.
2. The serial adder output is gated to the selected ST byte.
3. L2 and ABC are decremented by 1. If L2 was zero before stepping, an exit is made to the high-order zero extend routine. If L2 was not zero but ABC equalled zero, an exit is made to the source fetch routine.
4. If L2 or ABC is not equal to zero, cycle 1 is repeated.

High-Order Zero Extend Routine

An entry is made into this routine when the last source byte has been processed. The selected ST byte contains the high-order source digit in bits 4–7; bits 0–3 are zeros.

The following actions are performed upon entry into the routine:

1. STAT H is set.
2. The selected mark trigger is set.
3. L1 and STC are decremented by 1.
4. If L1 or STC equals zero before stepping, an exit is made to the destination store routine.

If L1 or STC is not zero, a 1-cycle loop is started, which:

1. Gates the serial adder output (zeros) to the selected ST byte.
2. Sets the mark trigger selected by the STC.
3. Decrements L1 and STC by 1.
4. Exits to the destination store routine when L1 or STC equals zero. (STAT H is set to cause re-entry into the high-order zeros routine after the destination is stored.)

Destination Store Routine

1. The contents of ST are stored into the destination field per the D-address.
2. A test is made for the end of the destination field. If the L1 count now equals all 1's, an exit is made to the common end-op sequence.
3. If L1 is not all 1's, D is decremented by 8.
4. If STAT H is set, the high-order zeros routine is resumed. If STAT H is not set, the byte processing loop is started at cycle 2.

Source Fetch Routine†

1. The source is requested from storage, and the IC is decremented by 8.
2. A word-overlap test is made by comparing the IC and D addresses.
3. If no word-overlap condition exists, the doubleword arriving from storage is gated to AB, and byte processing is resumed.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Transfer bits 4–7 of selected AB byte to bits 0–3 of selected ST byte.

† This routine is shared with the Pack instruction. Return to the appropriate microprogram is effected per STAT D.

2. Transfer bits 0–3 of selected AB byte to bits 4–7 of selected ST byte.
3. Update AB from ST, and repeat first step.
4. Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A flowchart of the execution of the Move with Offset instruction when a word-overlap condition exists is shown in Diagram 5-313, FEMDM. Basically, this microprogram specifies a 3-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

Cycle 1

This cycle is identical with cycle 1 in the not-word-overlap microprogram.

Cycle 2

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7).
2. The serial adder output is gated to the selected ST byte.

Cycle 3

1. If the STC is less than 4, the contents of S are transferred to A.
2. The contents of T are transferred to B via the parallel adder.
3. L2 and ABC are decremented by 1.
4. An exit is made to the high-order zeros routine if L2 was equal to zero before stepping. An exit is made to the source fetch routine if the ABC was equal to zero and L2 was not zero.
5. If no exit conditions exist, cycle 1 is repeated for the next byte.

High-Order Zero, Destination Store, and Source Fetch Routines

The high-order zero and destination store routines are the same as in the not-word-overlap sequence. The source fetch routine, however, is different.

Upon detecting a word-overlap condition, the source from main storage is not used. Instead, AB is updated from ST: if the STC is equal to 7, the contents of T are transferred to B; if the STC is not 7, the contents of S are transferred to A and the contents of T to B.

This section discusses the 32 logical instructions. The instructions use all five formats and operate on fixed- and variable-field length data. For a discussion of data formats, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.

GENERAL INITIALIZATION SEQUENCE

Before execution of SS logical instructions, a General Initialization Sequence (GIS) is performed (Diagram 5-401, FEMDM). The general function of the GIS is to set up initial conditions for the execution phase. These include:

1. Setting of STC and ABC. The STC is set to the rightmost first operand byte in ST, the byte to be processed first. Because the address of the rightmost byte is specified by D(21–23), the STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by transferring IC(21–23) to the ABC.
2. Transferring the first operand to ST during the first cycle of GIS.
3. Transferring the second operand to AB. An IC request for the second operand is issued on the first cycle of GIS; subsequently, GIS transfers the operand from the SDBO to AB.
4. Performing a word overlap test. For the purpose of this test, refer to "Word Overlap Condition", Section 4 of this Chapter.

At the completion of SS I-Fetch, a branch is made per the instruction op-code to the appropriate GIS microprogram. Note that, because of similarities in the GIS microprograms, the SS logical instructions are divided into two groups. One group consists of the Translate and Translate and Test instructions; the remaining SS instructions form the second group.

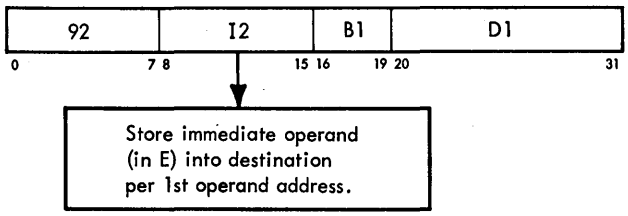
MOVE

Four logical move instructions are available:

1. Move, MVI, SI format. Places an immediate operand into the first operand location.
2. Move, MVC, SS format. Places the second operand into the first operand location.
3. Move Numerics, MVN, SS format. Places the numerics of the second operand bytes into the corresponding positions of the first operand bytes.
4. Move Zones, MVZ, SS format. Places the zones of the second operand bytes into the corresponding positions of the first operand bytes.

Move, MVI (92)

- Place immediate operand (I2 of instruction) into 1st operand location (in storage).
- SI format:

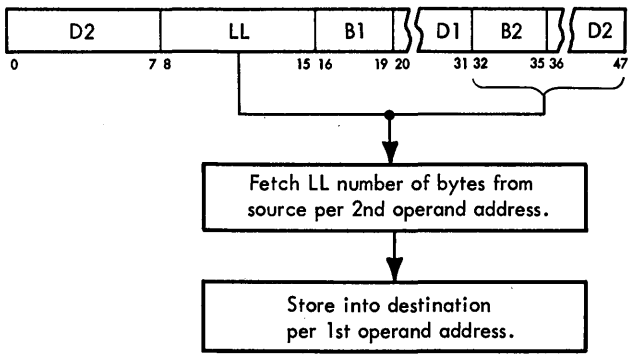


- Conditions at start of execution:
  First 16 bits of instruction, containing immediate operand, are in E.
  Main storage request for 1st operand has been issued per D.

The Move, MVI, instruction places the immediate operand into the first operand location. The immediate operand (I2 of instruction) is in E.

Move, MVC (D2)

- Place 2nd operand (in storage) into 1st operand location (in storage).
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC is set to select byte to be routed through serial adder.
  First 16 bits of instruction are in E.

- Move operation can be high or low speed.
- Three separate microprograms are provided:
  High-speed move.
  Word overlap.
  Low-speed move.

Three separate sequences are provided for the MVC instruction. The high-speed move sequence is used when it is possible to transfer a doubleword of data at a time. This condition exists when the high-order bytes of the source and destination are specified on doubleword boundaries and a full doubleword of data remains to be processed; i.e., both the ABC and STC are equal to zero, and the LL count is greater than 6. The word-overlap sequence is used when a word-overlap condition exists. The second operand in AB is updated after each AB byte is processed. The low-speed move sequence is used when the high-speed or word-overlap condition does not exist. (The high-speed and word-overlap conditions are detected in the GIS of the instruction.)

1. Low-Speed Move Sequence
   This sequence is basically a 1-cycle operation in which the AB byte selected by the ABC is transferred through the serial adder to the ST byte selected by the STC, and the mark trigger selected by the STC is set.

   The STC and ABC are incremented, the LL count in E(8–15) is decremented, and the cycle is repeated for the next byte, unless an exit condition exists.
2. Word-Overlap Move Sequence
   This sequence is a 2-cycle sequence in which the first cycle transfers the byte, selected by the ABC, to the ST byte selected by the STC. The second cycle updates the source operand in AB by transferring S to A, or T to B, as determined by the value of the STC. The mark trigger selected by the STC is set. The STC and ABC are incremented, the LL count is decremented, and the sequence is repeated for the next byte, unless an exit condition exists.
3. High-Speed Move Sequence
   This routine is entered from the GIS or from the low-speed move routine.
   a. When the entrance is made from the GIS, the source operand has been transferred to ST. The contents of ST are stored by setting mark triggers 0–7 and issuing a storage request per D.
   b. The LL count in E(8–15) is decremented by 8 via the parallel adder and is then tested for all 1's. If this condition exists, an end-op sequence is started. If no end-op condition exists, the IC is incremented by 8 via the parallel adder and a source fetch request is given.
   c. When the entrance is made from the low-speed routine, D is incremented by 8 and the source doubleword from main storage is gated to both AB and ST. If at least 8 bytes remain to be processed, as determined by a ROS branch on LL count being greater than 6, the high-speed move sequence is repeated (starting at step a). If fewer than 8 bytes remain to be processed, the low-speed move sequence is started to process the remaining data.

Exit is made from the low-speed or word-overlap move routines is one of the following conditions exists: (1) LL = 0, or STC = 7 and ABC ≠ 7; (2) LL = 0, or STC = 7 and ABC = 7; (3) only ABC = 7. A separate sequence is entered for each of these conditions, as explained below:

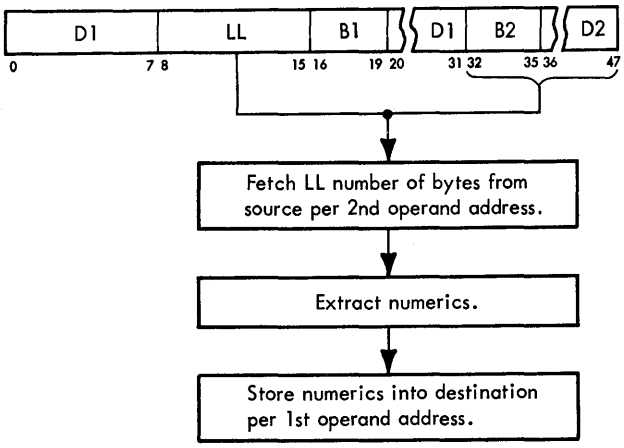
1. LL = 0, or STC = 7 and ABC ≠ 7
   A destination store is initiated, and a test for an end-op condition is made. If the LL count now equals all 1's, an entry is made into a common end-op sequence. If an end-op condition does not exist, D is incremented by 8 via the parallel adder and the low-speed move sequence is continued.
2. LL = 0, or STC = 7 and ABC = 7
   A destination store is initiated, and a test for end-op is made (LL = all 1's). A further test for a high-speed move condition is made. If at this time the LL count is 7 or greater, the IC and D are incremented by 8, a source fetch is initiated, and an entry is made into the high-speed move sequence. If neither an end-op nor a high-speed move condition exists, D is incremented by 8 and a common source fetch routine is entered which increments the IC by 8, fetches the next doubleword of source to AB, and tests for a word-overlap condition. Because there is no word-overlap at this time (ABC = STC), the low-speed move sequence is continued.
3. ABC = 7
   The IC is incremented by 8 through the parallel adder, and a fetch request is given to fetch the next doubleword of source operand. The common source fetch sequence is entered, which tests for word overlap. In this case, word overlap may exist: if it is detected, the source operand from main storage is not gated to AB, but instead ST is gated to AB and a branch is made to the move-word-overlap sequence. If no word overlap exists, the low-speed move sequence is continued after the source operand from main storage is gated to AB.

The common end-op routine is entered when the LL field has been decremented to zero. This routine restores the instruction address from the LSWR to the IC and resets STAT G (because it may have been used during the GIS).

Move Numerics, MVN (D1)

- Place numeric portion (low-order 4 bits) of each byte of 2nd operand (in storage) into low-order 4 bits of corresponding byte of 1st operand (in storage).

- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- Separate microprogram is used for word overlap.

The MVN instruction is executed as follows:

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7).
2. Bits 0–3 of the selected ST byte are gated to SAB(0–3).
3. Adder output is gated back to the selected ST byte.

Data is processed one byte at a time, and the fields may overlap in any way. Separate sequences are used for the not-word-overlap and the word-overlap conditions:

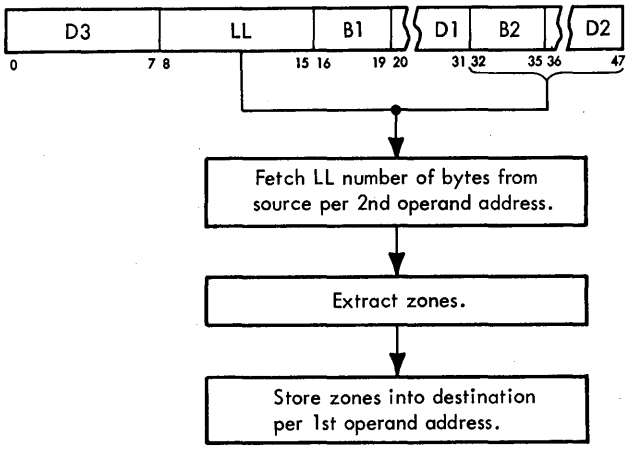
1. Not-Word-Overlap Sequence
   This sequence consists of a 1-cycle loop with an exit when LL = 0, STC = 7, or ABC = 7. As each byte is processed, the corresponding mark trigger is set per the STC; ABC and STC are incremented by 1 and LL is decremented by 1.
2. Word-Overlap Sequence
   This sequence consists of a 2-cycle loop with an exit when ABC or STC = 7, or when LL = 0.
   a. Cycle 1
      Numeric (bits 4–7) is moved from AB to ST.
   b. Cycle 2
      The contents of S are transferred to A, or the contents of T are transferred to B as determined by the STC value. The mark trigger selected by the STC is set; STC and ABC are incremented by 1, and LL is decremented by 1.

An exit from the byte processing sequence is made when LL = 0, STC = 7, or ABC = 7. A separate sequence is entered for each of these conditions, as explained below:

1. LL = 0
   The contents of ST are stored per D into the destination field. The common end-op sequence is started.
2. STC = 7
   The common destination store-fetch routine is started. If the ABC also equals 7, STAT D is set to cause a source fetch before resuming the byte processing loop.
3. ABC = 7
   The common source fetch routine is started, which includes a word-overlap test, which causes the appropriate instruction word-overlap or not-word-overlap loop to be continued.

Move Zones, MVZ (D3)

- Place zone portion (high-order 4 bits) of each byte of 2nd operand (in storage) into high-order 4 bits of corresponding byte of 1st operand (in storage).
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  STC and ABC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- Separate microprogram is used for word overlap.

The MVZ instruction specifies the following actions:

1. Bits 0–3 of the selected AB byte are gated to SAA(0–3).
2. Bits 4–7 of the selected ST byte are gated to SAB(4–7).
3. The adder output is gated back to the selected ST byte.

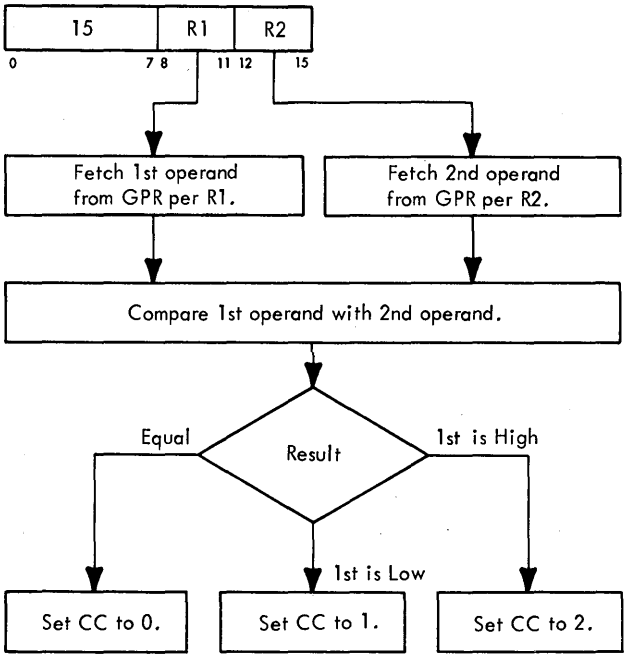
Except for the above actions, the byte processing sequence is the same as that for the MVN instruction.

COMPARE

Four Compare Logical instructions are provided, in the RR, RX, SI, and SS formats. Comparison is binary, and all codes are valid. Operation is terminated when an inequality is found.

Compare Logical, CLR (15)

- Binarily compare 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and set CC according to result.
- RR format:



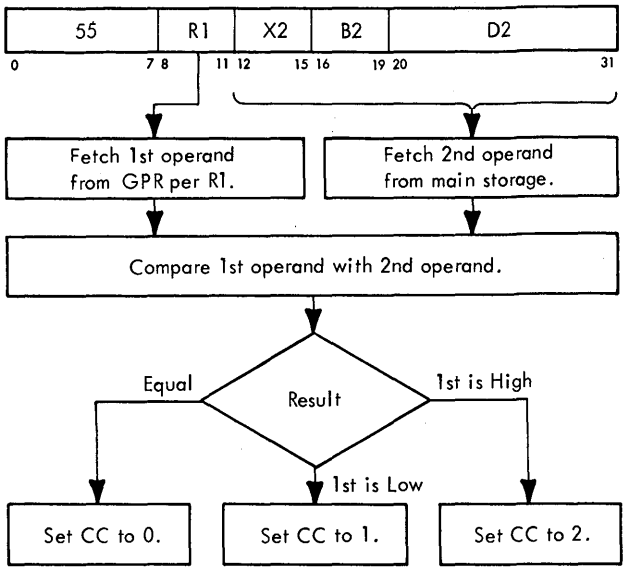
- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.

- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CLR, instruction, which is in the RR format, compares the first operand with the second operand. Comparison in binary, and is performed left to right, byte by byte. The CC is set according to the result.

Compare Logical, CL (55)

- Binarily compare 1st operand (in GPR, per R1) with 2nd operand (in storage) and set CC according to result.
- RX format:



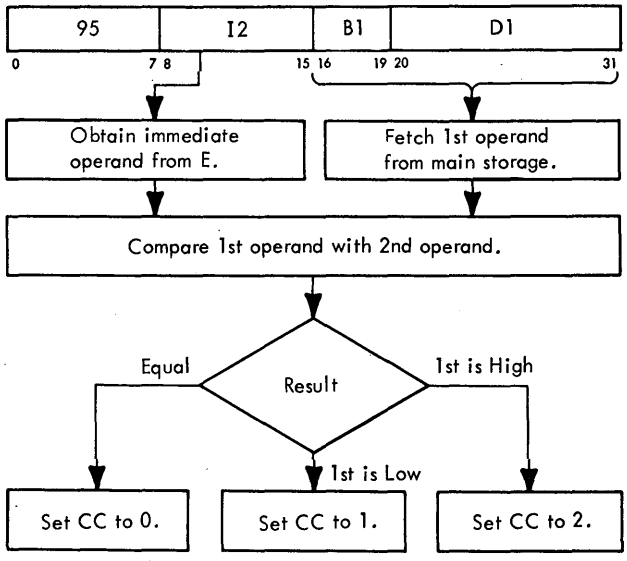
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CL, instruction, which is in the RX format, compares the first operand with the second operand. Comparison is binary, and is performed left to right, byte by byte. The CC is set according to the result.

Compare Logical, CLI (95)

- Binarily compare 1st operand (in storage) with immediate operand (I2 of instruction) and set CC according to result.
- SI format:

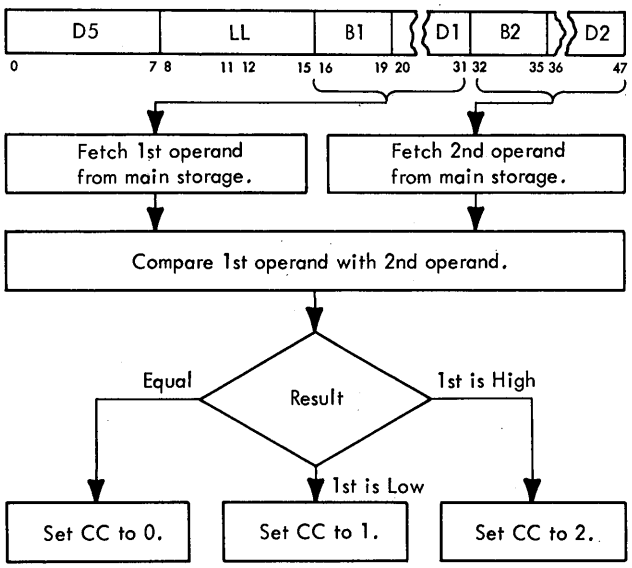


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  1st 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CLI, instruction, which is in the SI format, compares the first operand with the immediate second operand. Comparison is binary, and is performed left to right. The CC is set according to the result.

Compare Logical, CLC (D5)

- Binarily compare 1st operand (in storage) with 2nd operand (in storage) and set CC according to result.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.
- Because results of operation are not stored into main storage, no special action is required during word overlap.

The CLC instruction is sequenced as follows:

1. The selected AB byte is gated complement to the serial adder with a hot carry to bit 7.
2. The selected ST byte is gated true to the serial adder.
3. The serial adder carry is saved in STAT H.
4. STAT A is set if a nonzero result byte is detected.
5. As each byte is processed, the LL count is decremented and the ABC and STC are incremented.
6. The above routine is continued until a nonzero result is detected in the serial adder, or until the LL count is stepped to zero, with exits for operand fetches when the STC or ABC is stepped to 7.
7. If an exit is made because a nonzero byte is detected, one additional byte will have been gated to the serial adder before the exit is made via the ROS branch. Therefore, STAT H will reflect the carry of the nonzero result byte plus 1. Because STAT H is used to determine the setting of the CC, it is set or reset per the carry of the first nonzero byte encountered.
8. The common end-op routine is used, which sets the CC per the following hardware conditions:

Hardware Conditions	CC Setting
STAT A is reset and equal compare	0
STAT A is set and STAT H is reset	1
STAT A and STAT H are set	2

AND

The AND instruction mixes two operands on a logical AND basis. An AND operation is defined as follows: if both operand bits are 1's, the resulting bit is 1; otherwise, the result is a 0. The following example illustrates the AND'ing of two bytes:

Bit positions	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	1	0	0	0	1	0	0	0

Note that only in bit positions 0 and 4 are both operand bits set to 1. Therefore, only bits 0 and 4 of the result are set to 1.

A simplified data flow path for AND, OR, and Exclusive-OR instructions is shown in Figure 3-25. All logical AND's, OR's, and Exclusive-OR's are performed in the serial adder, a byte at a time. The first operand is placed into ST and the second operand is placed into AB. Upon completion of the instruction, the result is placed into ST.

The sequencing of individual bytes from ST and AB through the serial adder is controlled by the STC and the ABC. In the fixed-format RR and RX instructions, the STC and ABC are preset to 4 and incremented to 7 as the four data bytes from ST and AB are routed through the serial adder. For SS instructions, the STC and ABC are preset to values that point to the bytes where the data starts in their

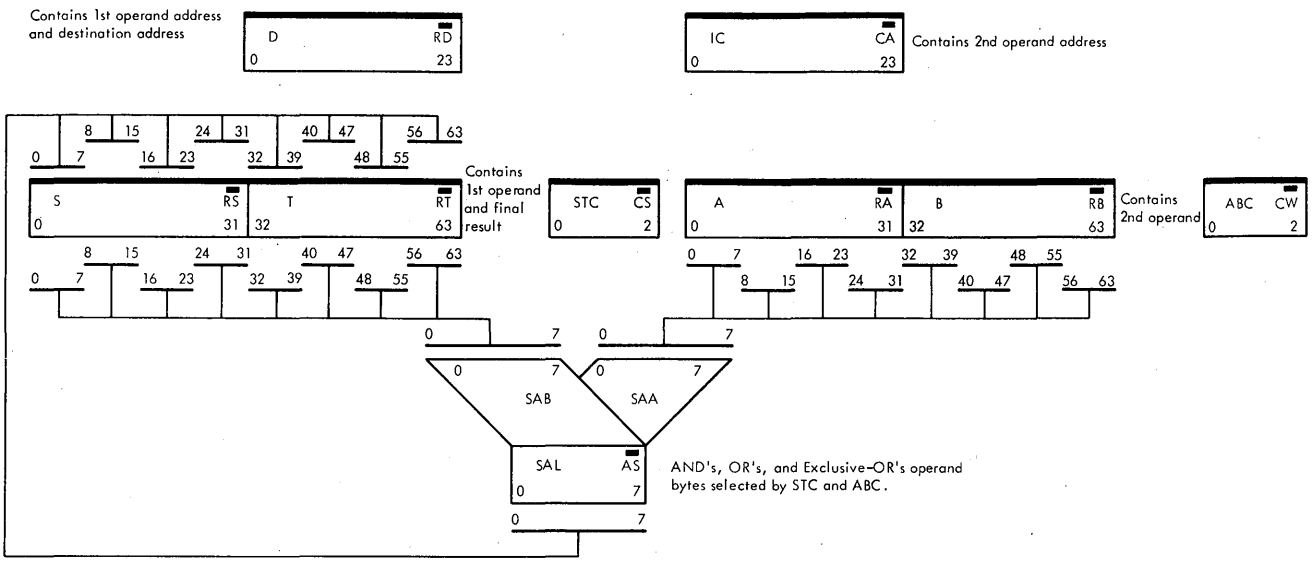
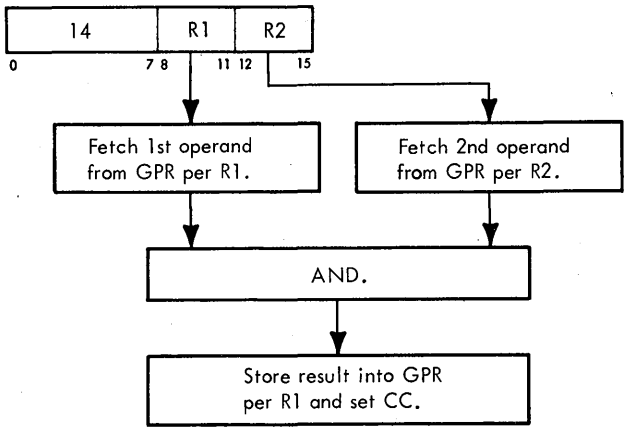


Figure 3-25. Simplified Data Flow for AND, OR, and Exclusive-OR Instructions

respective registers. For SI instructions, in which the second operand is one byte long (I2 field), the ABC points to the second operand location in AB, and the STC points to the first operand in ST.

AND, NR (14)

- AND 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:

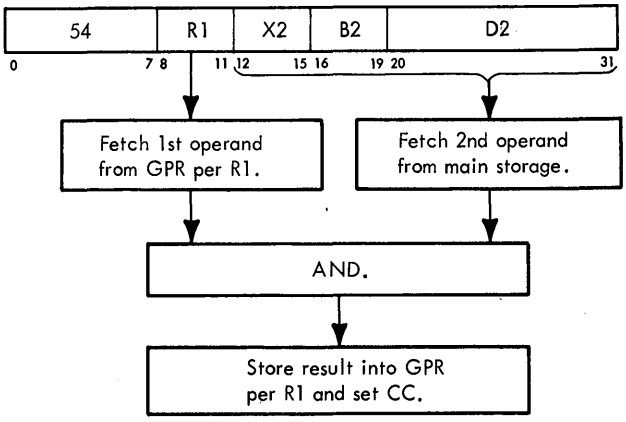


- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, NR, instruction, which is in the RR format, AND's the first operand with the second operand. The AND function is applied left to right, byte by byte.

AND, N (54)

- AND 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:



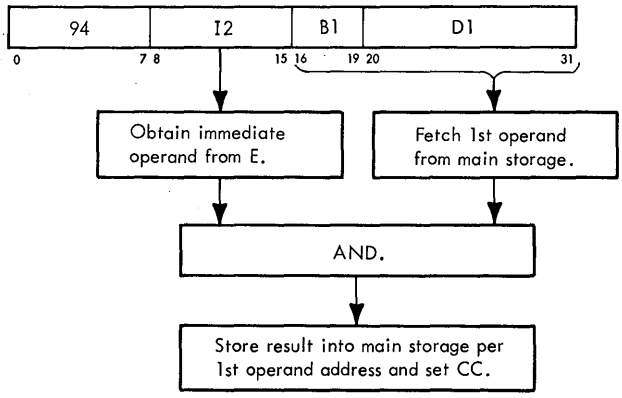
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, N, instruction, which is in the RX format, AND's the first operand with the second operand from main storage. The AND function is applied left to right, byte by byte.

AND, NI (94)

- AND immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

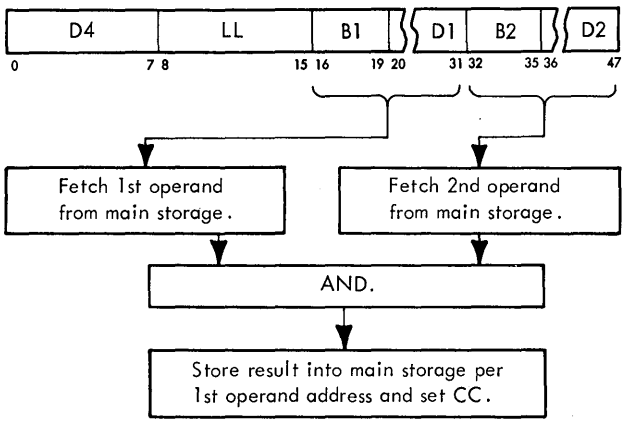


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, NI, instruction, which is in the SI format, AND's the first operand with the immediate second operand. The AND function is applied left to right.

AND, NC (D4)

- AND 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

The NC instruction specifies the following actions:
1. The selected AB byte is gated to SAA(0–7).
2. The selected ST byte is gated to SAB(0–7).
3. Each AB and ST bit is combined using the serial adder AND function.
4. The adder output is gated back to ST. (STAT A is set if the result byte is not zero.)

Except for the above actions, the byte processing sequence is the same as for the Move Numerics instruction.

OR

The OR instruction mixes two operands on a logical OR basis. An OR operation is defined as follows: if either operand bit is a 1, the resulting bit is a 1: otherwise, the

result is a 0. The following example illustrates the OR'ing of two bytes.

Bit positions	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	1	0	1	1	1	1	1	0

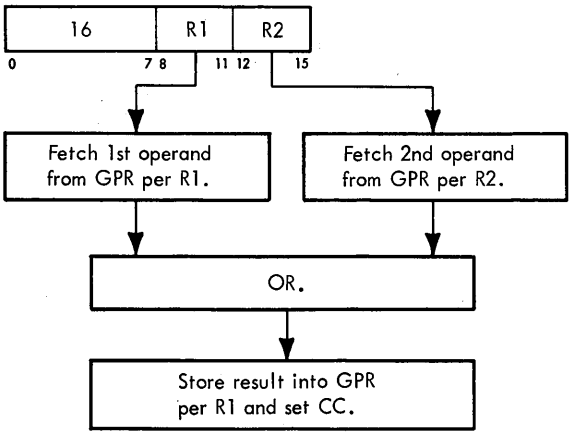
Note that only in bit positions 1 and 7 is neither bit a 1. Thus, only bits 1 and 7 of the result are set to 0, and the remaining bits are set to 1.

The sequencing of operands through the serial adder is similar to the sequencing of the AND instructions. The major difference is that the serial adder applies the OR function.

The OR operation may be executed by an instruction in the RR, RX, SI, or SS format.

OR, OR (16)

- OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:

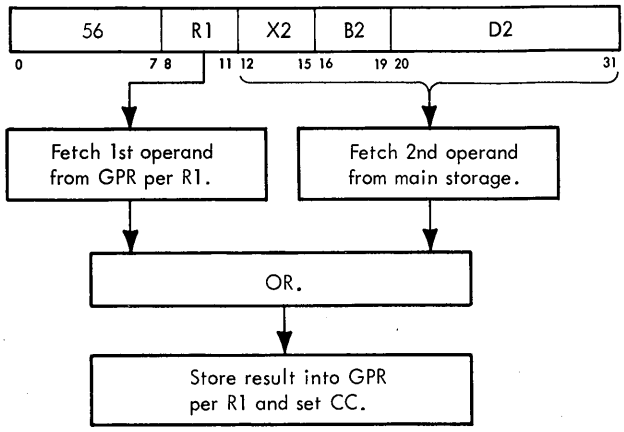


- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, OR, instruction, which is in the RR format, OR's the first operand with the second operand. The OR function is applied left to right, byte by byte.

OR, O (56)

- OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:

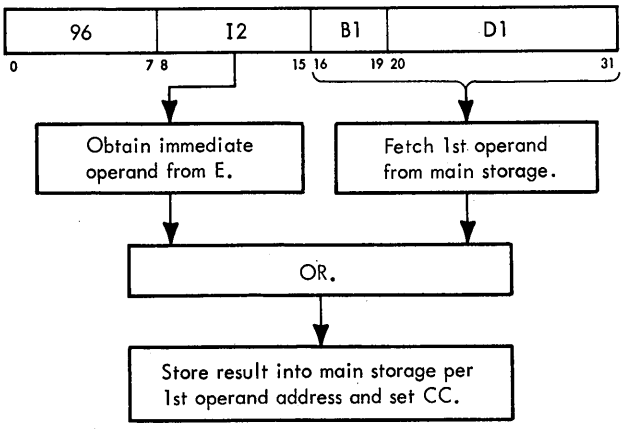


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, O, instruction, which is in the RX format, OR's the first operand with the second operand from main storage. The OR function is applied left to right, byte by byte.

OR, OI (96)

- OR immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

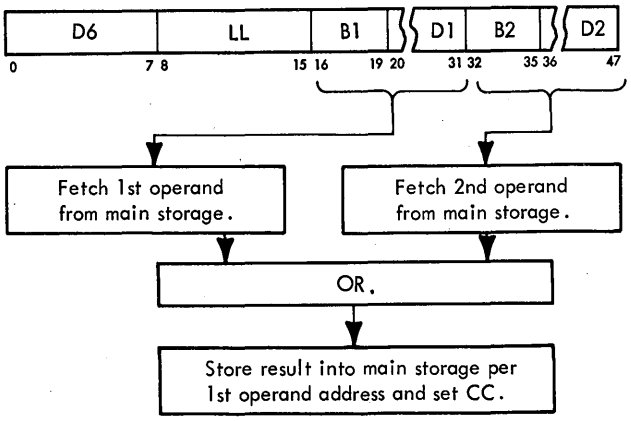


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, OI, instruction, which is in the SI format, OR's the first operand with the immediate second operand. The OR function is applied left to right.

OR, OC (D6)

- OR 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

The OC instruction specifies the following actions:

1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per the STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the Move Numerics instruction.

EXCLUSIVE-OR

The Exclusive-OR instruction mixes two operands on a logical Exclusive-OR basis. An Exclusive-OR operation is defined as follows: if one and only one of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is a 0. The following example illustrates the Exclusive-OR'ing of two bytes.

Bit position	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	0	0	1	1	0	1	1	0

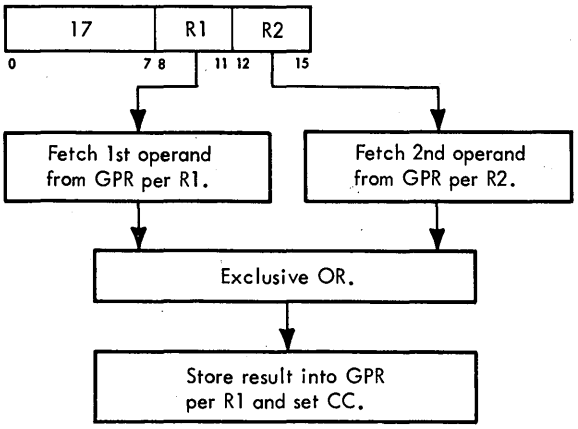
Note that in bit positions 2, 3, 5, and 6 one and only one of the operand bits is a 1, and that the corresponding bit positions of the result are set to 1. In bit position 0, both operand bits are 1 and the corresponding result bit is 0. In bit position 1, both bits are 0 and the result is 0.

The sequencing of operands through the serial adder is similar to the sequencing of the AND instructions. The major difference is that the serial adder applies the Exclusive-OR function.

The Exclusive-OR operation may be executed by an instruction in the RR, RX, SI, or SS format.

Exclusive-OR, XR (17)

- Exclusive-OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:



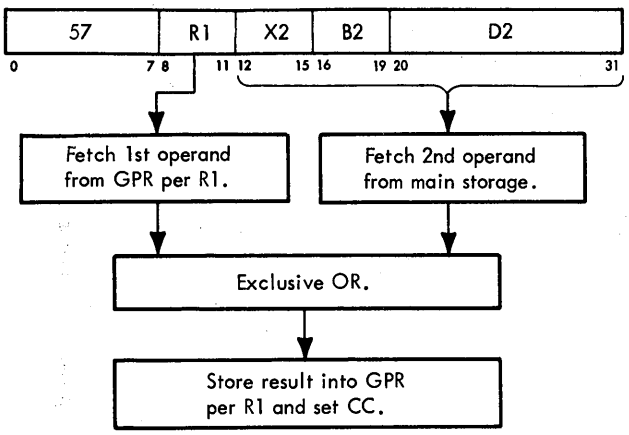
- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, XR, instruction, which is in the RR format, exclusive-OR's the first operand with the second operand. The exclusive-OR function is applied left to right, byte by byte.

Exclusive-OR, X (57)

- Exclusive-OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:

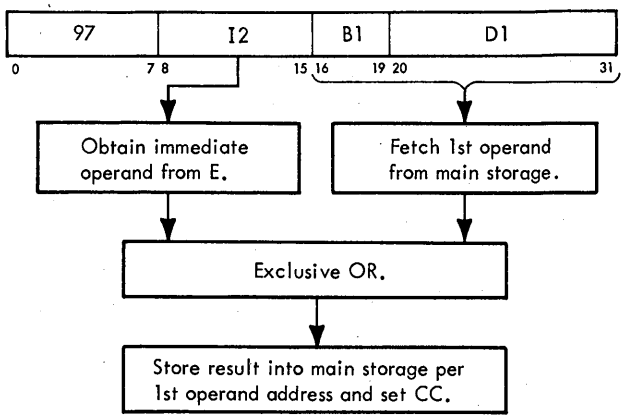


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, X, instruction, which is in the RX format, exclusive-OR's the first operand with the second operand from main storage. The exclusive-OR function is applied left to right, byte by byte.

Exclusive-OR, XI (97)

- Exclusive-OR immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

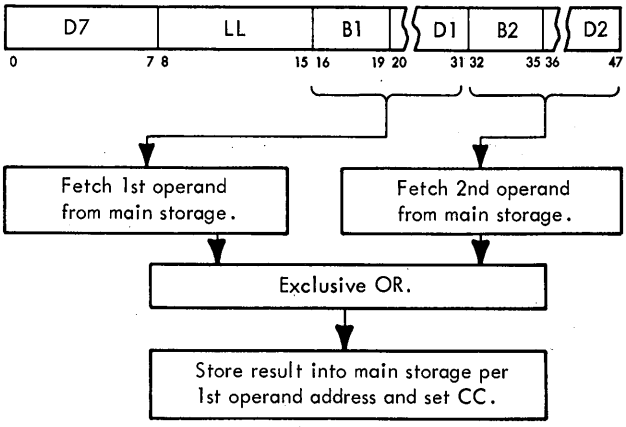


- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, XI, instruction, which is in the SI format, exclusive-OR's the first operand with the immediate second operand. The exclusive-OR function is applied left to right.

Exclusive-OR, XC (D7)

- Exclusive-OR 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:



- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

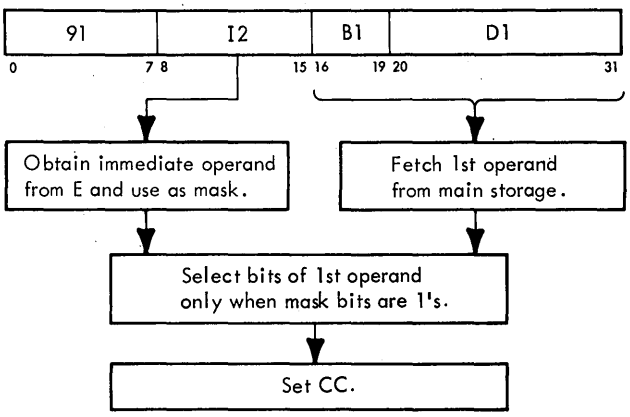
The XC instruction specifies the following actions:

1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder Exclusive-OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per the STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the Move Numerics instruction.

TEST UNDER MASK, TM (91)

- Set CC according to state of 1st operand bits (in storage) selected by mask bits (I2 of instruction).
- SI format:



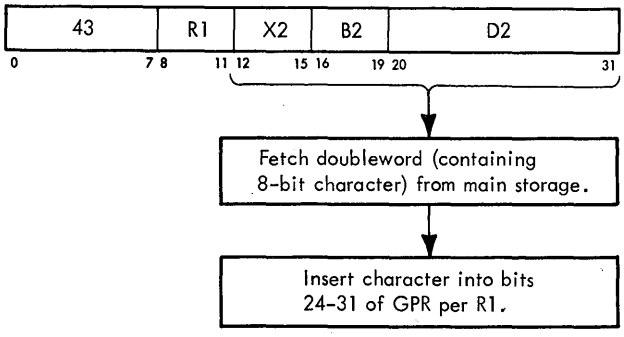
- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Selected bits all zero; mask is all zero: CC = 0.
  Selected bits mixed zero and 1: CC = 1.
  Selected bits all 1: CC = 3.
- Storage contents are not changed.

The byte of immediate data, I2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in main storage specified by the first operand address.

A mask bit of 1 indicates that the storage bit is selected. When the mask bit is 0, the storage bit is ignored. When all storage bits thus selected are zero, the CC is made 0. The CC is also made 0 when the mask is all-zero. When the selected bits are all-1, the CC is made 3; otherwise, the CC is made 1. The character in storage is not changed.

INSERT CHARACTER, IC (43)

- Insert 2nd operand (byte; in storage) into bits 24–31 of 1st operand location (in GPR, per R1).
- RX format:



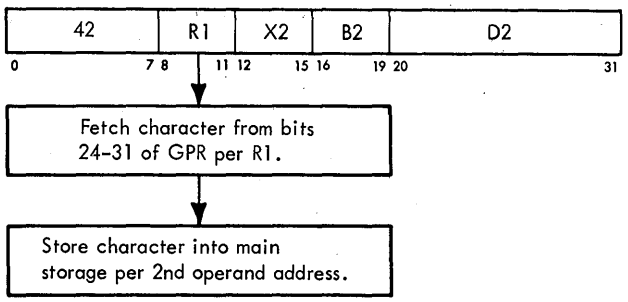
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Insert Character instruction, which is in the RX format, inserts the byte of second operand into bits 24–31 of the GPR specified by R1. The remaining bits in the GPR are unchanged.

STORE CHARACTER, STC (42)

- Store bits 24–31 of 1st operand (in GPR, per R1) into 2nd operand location (in storage).

- RX format:

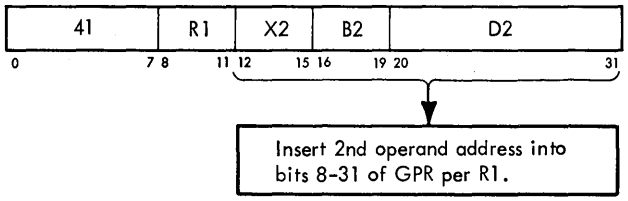


- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Store Character instruction, which is in the RX format, stores the byte (bits 24–31) of first operand into main storage per the second operand address.

LOAD ADDRESS, LA (41)

- Insert 2nd operand address into bits 8–31 of GPR specified by R1.
- RX format:



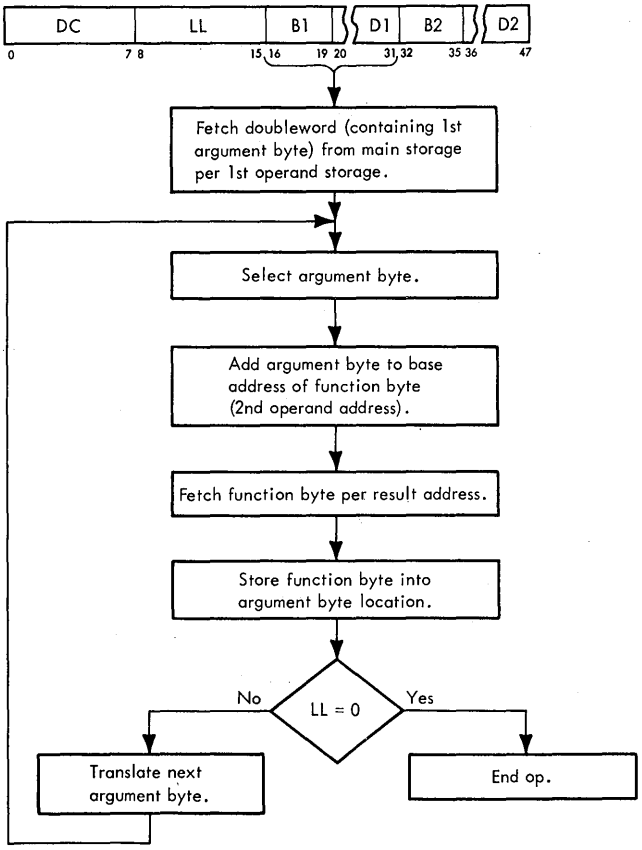
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The address specified by the X2, B2 and D2 fields is inserted into bits 8–31 of the GPR specified by R1; bits 0–7 are made 0's. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 24th bit are ignored. The same GPR may be specified by the R1, X2, and B2 instruction field, except that GPR0 can be specified only by the R1 field. In this manner, it is possible to increment the low-order 24 bits of a GPR, other than 0, by the contents of the D2 field of the instruction. The GPR to be incremented should be specified by R1 and by either X2 (with B2 set to zero) or B2 (with X2 set to zero).

TRANSLATE TR (DC)

- Add 1st operand byte (argument; in storage) to effective 2nd operand address, use result as storage address, and place function byte from resulting storage address into corresponding 1st operand byte location.
- SS format:



- Conditions at end of GIS:
  1st operand (destination) is in ST.
  Destination address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.

The Translate instruction selects the first operand bytes for translation one byte at a time, proceeding from left to right. Each argument byte is added to the entire initial address, the second operand address in the low-order bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte. All

data is valid. The operation proceeds until the first operand field is exhausted. The table is not altered unless an overlap occurs.

At the start of the execution sequence, the first operand has been fetched to ST. A request per the IC has been made for the second operand, but this doubleword from main storage is not used.

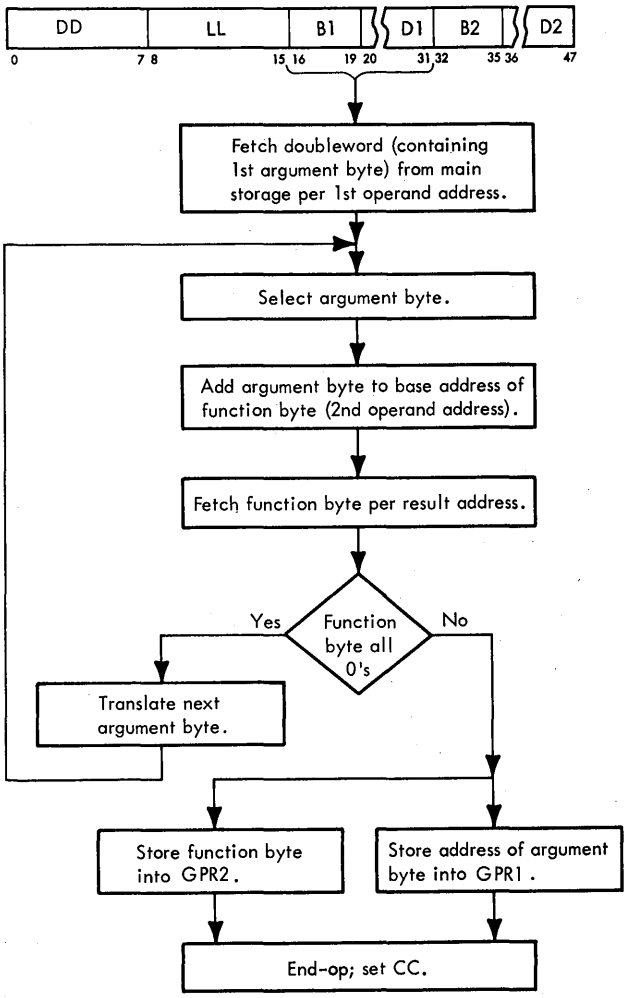
The execution sequence is as follows:

1. The selected ST byte is saved in F. The contents of T are saved in B. (The contents of the IC are saved in A.)
2. T is cleared, the STC is set to 111, and the contents of F (selected destination byte) are placed into T(56–63) via the serial adder.
3. The contents of T are added to the contents of the IC in the parallel adder, and the result is gated back to the IC.
4. A request for the second operand is issued per the IC.
5. The ABC is set per IC(21–23), and the STC is set per D(21–23).
6. The original source address is restored to the IC from A. The destination word is restored to T from B.
7. A word-overlap test was made before the source address was restored to the IC. If no word overlap exists, the table doubleword fetched from main storage is gated to AB. If word overlap is detected, the contents of S are transferred to A (B is already identical with T) and the doubleword from main storage is not used.
8. The selected AB byte is gated via the serial adder to the selected ST byte, and the selected mark trigger is set. The STC and D are incremented by 1. The LL count is decremented by 1.
9. Unless the STC was 7 or LL was zero before stepping, the sequence is repeated for the next destination byte.
10. If LL was zero, the contents of ST are stored and the common end-op sequence is started.
11. If the STC was 7 and LL not equal to zero, the contents of ST are stored and the next destination word is fetched by the common destination fetch sequence, after which the translate sequence is resumed.

TRANSLATE AND TEST, TRT (DD)

- Add 1st operand byte (argument; in storage) to effective 2nd operand address, use result as storage address, and test function byte from resulting storage address. If 0, translate and test next argument byte; if non-0, complete operation by inserting related argument address into GPR1 and function byte into GPR2.

- SS format:



- Conditions at end of GIS:
  1st operand (destination) is in ST.
  Destination address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.

- CC setting:
  All function bytes are zero: CC = 0.
  Nonzero function byte encountered before operand is exhausted: CC = 1.
  Last function byte is nonzero: CC = 2.

The Translate and Test instruction fetches the function bytes in the same manner as the Translate instruction. Each function byte retrieved from the table is inspected for an all-zero combination.

When the function byte is zero, the operation proceeds with the next operand byte. When the first operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting the CC to 0. The contents of GPR's 1 and 2 remain unchanged.

When the function byte is nonzero, the related argument address is inserted into the low-order 24 bits of GPR1. This address indicates the argument last translated. The high-order eight bits of GPR1 remain unchanged. The function byte is inserted into the low-order eight bits of GPR2. Bits 0–23 of GPR2 remain unchanged. The CC is set to 1 when one or more argument bytes have not been translated. The CC is set to 2 if the last function byte is nonzero.

The following abbreviations are used in this discussion of the Translate and Test execution sequence:

DX: first byte in series of destination bytes.

T(DX): table byte specified by DX.

DX + 1: second byte in series of destination bytes.

T(DX + 1): table byte specified by DX + 1.

DX + 2: third byte in series of destination bytes.

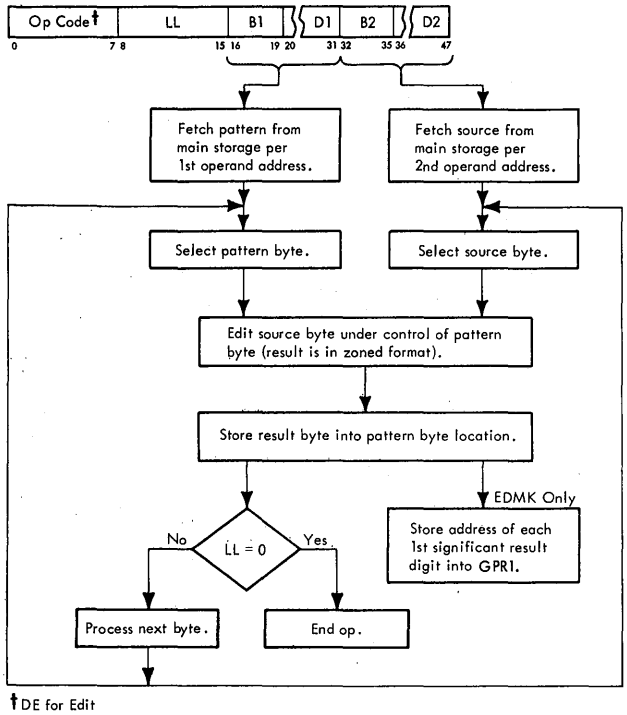
The Translate and Test instruction uses the following execution sequence:

1. First Byte Sequence
   a. The selected ST byte is saved in F.
   b. The contents of ST are transferred to AB.
   c. The STC is set to 3, and the contents of F (DX) are gated, via the serial adder, to byte 3 in S.
   d. Bytes 0, 1, and 2 in S are cleared by gating the contents of SAL to ST and successively decrementing the STC by 1.
   e. The ABC is set per D(21–23), and the STC is set to 3.
   f. The DX in S is added to the contents of the IC, and an IC request is made for T(DX).
   g. A branch per STAT G is made to the T(DX + 1) address generation routine. (STAT G is used to indicate that a table byte has been fetched and is ready for test.)
2. T(DX + 1) Address Generation
   a. The ABC is incremented by 1.
   b. DX is transferred from S to T.
   c. STAT G is set.
   d. The selected AB byte (DX + 1) is gated via the serial adder to byte 3 in S.
   e. The STC is set per IC(21–23).
   f. The T(DX) ingate and T(DX + 1) fetch sequence is started.
3. T(DX) Ingate and T(DX + 1) Fetch Sequence
   a. The table word which contains byte T(DX) is available from main storage, and either the left- or right-half word is gated to T as determined by IC(21). STC(0) is set to 1 to select the correct byte in T. Simultaneously, the contents of T(DX) are subtracted from the contents of IC to restore the table base address.
   b. If LL equals zero, an exit is made to the T(DX) test sequence.
   c. If LL is not zero, DX + 1 (in S) is added to the contents of IC and a fetch request is made for T(DX + 1).
   d. A branch per STAT G starts the T(DX) test sequence.
4. T(DX) Test Sequence and T(DX + 2) Address Generation
   a. The selected byte in T, T(DX), is gated to the serial adder for zero detection and is saved in F.
   b. STAT H is set if the ABC equals zero.
   c. DX + 1 is transferred from S to T.
   d. The STC is set to 3, and the ABC is incremented by 1 (selecting byte to DX + 2).
   e. An exit is made to the LS mark sequence (step 6) if a nonzero result is detected in the serial adder.
   f. An exit is made to the common end-op sequence if the serial adder result is zero and the LL count is zero.
   g. The LL count is decremented and the address in D is incremented by 1.
   h. If STAT H is set, an exit is made to the destination fetch routine.
   i. If no exit conditions are detected, the selected AB byte (DX + 2) is gated via the serial adder to S, and the STC is set per IC(21–23).
   j. The T(DX) ingating and T(DX + 1) fetch sequence is started. The table byte previously referred to as T(DX) has been tested. The table byte previously referred to as T(DX + 1) is now considered T(DX), and the processing loop is resumed.
5. Destination Fetch Routine
   a. Before entering this routine, the ABC has been stepped from 7 to 0 and a fetch request was made for a table byte using byte 0 of the present destination word to generate the table byte address. Because this was an erroneous address, the resulting word from main storage is not used.
   b. STAT's G and H are reset, and the IC is restored to the table base address by subtracting DX + 1.
   c. A fetch request is made per the D-address. The requested doubleword is gated to AB.
   d. The ABC was previously stepped from 0 to 1. It is now decremented to select byte 0 of the new destination doubleword (considered byte DX).
   e. The selected AB byte (DX) is gated via the serial adder to byte 3 in S.
   f. The DX (in S) is added to the contents of IC, and a fetch request is made for T(DX).
   g. Because STAT G is reset, the T(DX + 1) address generation routine is started.

6. LS Mark Sequence
   a. This routine is entered when a nonzero table byte is detected in the serial adder or when the LL count equals zero. (The last table byte tested is in F.)
   b. If the table byte was nonzero, STAT G is reset.
   c. E(8–15) is cleared and used for LAR addressing.
   d. GPR1 is accessed per E(8–15) + 1 and its contents transferred to T.
   e. The STC is incremented to 4, and byte 4 of ST is gated via the serial adder back to ST. Simultaneously, the contents of D are gated to T via the parallel adder.
   f. The contents of T are stored into GPR1; E(8–11) is incremented twice, and the STC is set to 7.
   g. The contents of GPR2 are transferred to T. The contents of F are gated via the serial adder to T(56–63), and the contents of T are stored into GPR2.
   h. STAT A is set if the byte in F was not zero.
   i. The common end-op sequence is started, which sets the CC per STAT's A and G.

EDIT AND EDIT AND MARK, ED AND EDMK (DE AND DF)

- Edit: change format of source (2nd operand; in storage) from packed to zoned, edit source under control of pattern (1st operand; in storage), and place result into 1st operand location.
- Edit and Mark: same as Edit, but in addition place location of each 1st significant digit into GPR1.
- SS format:



†DE for Edit
DF for Edit and Mark

- Conditions at end of GIS:
  Pattern (destination operand) is in ST.
  Pattern address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.

The Edit instruction changes the format of the source (second operand) from packed to zoned, edits the source under control of a pattern (first operand), and places the result into the first operand location. The Edit and Mark instruction performs the same functions and, in addition, places the location of each first significant digit into GPR1. Both instructions are in the SS format, and share a common ROS microprogram with an exit to a separate mark routine for the Edit and Mark instruction. Because the results of a word-overlap condition are unpredictable, no special action is taken when this condition occurs.

Introduction to Edit Operation

- Edit instruction is used to:
  Eliminate high-order zeros.
  Provide asterisk protection.
  Handle sign control (CR).
  Provide punctuation.
  Blank out an all-zero field.
  Protect decimal point by use of significance start character. (This character can also be used to retain high-order zeros when desired.)
  Edit multiple adjacent fields via field separator character.

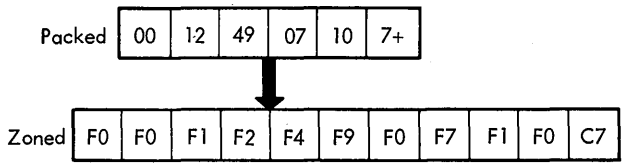
The edit operation is used to produce easy-to-read documents by inserting proper punctuation into a data record. The data to be edited (second operand) is called the "source" and must be in the packed BCD format. Consider the following source field:

00	12	49	07	10	7+

For the above field to be printed in a document, it must first be converted into the zoned format (USASCII-8 or EBCDIC). One function of the edit operation is to change the source field from packed to zoned format.† If changing

† Each time the digit from the source field replaces a digit select character, the four-bit digit has the proper EBCDIC or USASCII-8 zone bits inserted. PSW(12) determines whether the EBCDIC or USASCII-8 zone is inserted. For this discussion, it is assumed that the system is in EBCDIC mode.

from packed to zoned format were all that was necessary to produce a legible report, the Edit instruction would not be necessary, because the Unpack instruction would be sufficient. For instance, if the above packed BCD operand were changed to the EBCDIC zoned format, it would look like this:



If the above zoned BCD field were printed, it would look like this:

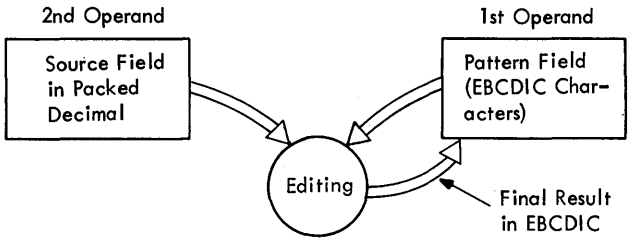
0 0 1 2 4 9 0 7 1 0 7 +

By examining the printed document, one could tell that it was a positive number with a low-order digit of 7. However, the printed document is still not legible. If, for instance, the number represents money, it would be desirable to obtain the following printed result:

$1,249,071.07

This result would require insertion of the commas and decimal points in the right place, as well as other editing. This is the main function of the edit operation.

The edit operation involves moving the source field (second operand) into the pattern field (first operand). The pattern field is initially made up of EBCDIC characters that control the editing. The final edited result replaces the pattern field:



As a rule, the second operand is shorter than the first because one source byte yields two result bytes.

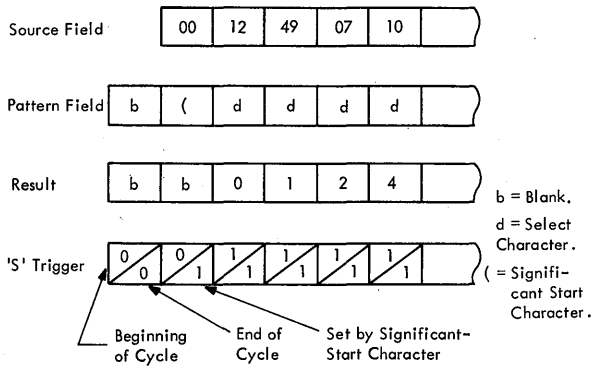
The characters in the pattern field determine the editing that takes place. The high-order (leftmost) character in the pattern field is known as the "fill" character. Any of the 256 possible EBCDIC combinations can be used as the fill character. In many edit operations, however, the fill character consists of an EBCDIC blank (0100 0000). The blank character (represented by "b" in the discussion that follows) is not printed out and facilitates programmed blanking of high-order zero fields.

Besides the fill character, three more control characters in the pattern field have special meaning: the digit select character, the significant start character, and the field separator character. These characters can appear anywhere in the pattern field.

For purposes of discussion, the digit select character is represented by "d." (The binary code for the digit select character is 0010 0000, or a hex 20.) When a digit select character is encountered in a pattern field, it is usually replaced with a digit from the source field. If the digit in the source field is a high-order zero, however, the digit select character is replaced by the fill character. By using a blank as the fill character, high-order zeros can be blanked out. If an asterisk is used as the fill character, asterisk protection for paychecks can be achieved.

Because the digit select character may be replaced by either a source digit or the fill character, the system needs some way of knowing which of the two to choose. This function is provided by a special control trigger, known as the 'S' trigger. When the 'S' trigger is set, it indicates that significant source digits are being processed. Consequently, the digit select characters in the pattern field are replaced with the digits from the source field. At the beginning of the edit operation, the 'S' trigger is always reset. As long as the 'S' trigger is reset, the digit select characters in the pattern field are replaced with the fill character.

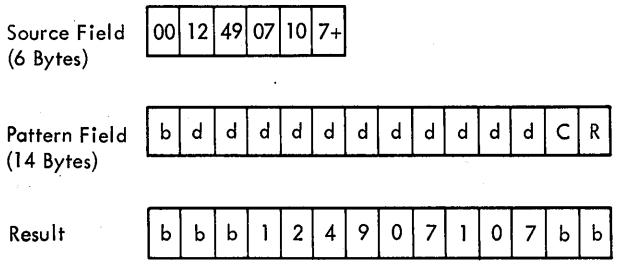
As stated previously, the 'S' trigger is set when a nonzero digit is detected in the source field. The 'S' trigger is also set if a significant start character is detected in the pattern field. The significant start character has a bit code of 0010 0001 (hex 21). In this discussion, the symbol for the left parenthesis is used to represent the significant start character. When a significant start character is detected in the pattern field, it is replaced by either a digit from the source field or the fill character. A typical edit operation using the b, d, and (characters is illustrated and explained below.



The edit operation begins by examining the fill character (which is b in the above case). If it is not a digit select or a

significant start character, it is left in place in the pattern field. Then, the next pattern character is examined. Because this is a significant start character, the next high-order source digit is examined. Because this source digit is zero and the 'S' trigger is reset (at this time), the significant start character is replaced with the fill character. However, the significant start character sets the 'S' trigger so that all subsequent source digits are significant. The remaining pattern characters in the above example are digit select characters, which are replaced with source digits.

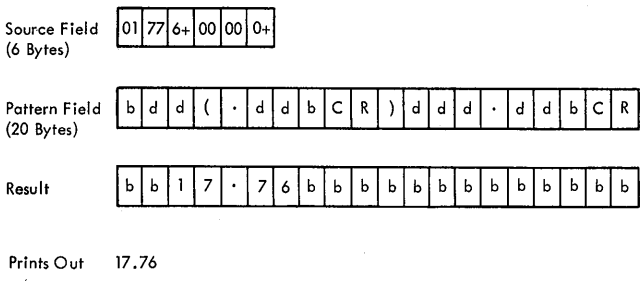
Once significance is started, the 'S' trigger remains set until the sign of the source operand is examined. If a plus sign is detected, the 'S' trigger is reset; if the source has a negative sign, the 'S' trigger remains set because the usual method of indicating a negative quantity in a printed report is with the letters "CR". The following example illustrates how the state of the 'S' trigger identifies the number as a positive or negative quantity:



When a pattern character is not one of the three special control characters and the 'S' trigger is set, the character is not changed. If the 'S' trigger is reset, the character is replaced by the fill character. Because detection of a positive sign resets the 'S' trigger, the remaining pattern characters (CR) are replaced by the fill character. If the sign of the source field had been minus, the 'S' trigger would have remained set and characters CR would have been left in the pattern field.

As stated previously, the 'S' trigger is reset when a plus sign is detected in the source field. The 'S' trigger is also reset if a field separator character is detected in the pattern field. The field separator character has a bit code of 0010 0010 (hex 22). In this discussion, the symbol for the right parenthesis represents the field separator character.

The field separator character is used when two or more packed BCD source fields are to be edited with one instruction into a single pattern field. The following edit example illustrates the use of the field separator character.



Note that after the field separator character resets the 'S' trigger, the source field does not contain any significant digits. As a result, the pattern characters are replaced by the fill character (blank).

Introduction to Edit and Mark Operation

The Edit and Mark operation is identical with that of the Edit instruction, except for the additional function of inserting a byte address into bits 8–31 of GPR1. The byte address is inserted each time the 'S' trigger is reset and a nonzero digit is inserted in the result field. The address is not inserted when significance is forced by the significant start character of the pattern. Bits 0–7 of GPR1 are not changed. The Edit and Mark instruction facilitates the programming of floating currency-symbol insertion. The character address inserted into GPR1 is 1 more than the address where a floating currency-sign would be inserted. (The Branch on Count instruction, with zero in the R2 field, may be used to reduce the inserted address by 1.)

The character address is not stored when significance is forced. Therefore, the address of the character following the significant start character should be placed into GPR1 before the Edit and Mark instruction is executed.

When a single instruction is used to edit several numbers, the address of the first significant digit of each number is inserted into GPR1. Only the last address will be available after the instruction is completed.

General Data Handling

Special circuits are packaged in the serial adder for use in the Edit and Edit and Mark instructions. These circuits consist of:

1. A decoder of serial adder bus B (SBB) to detect a digit select, significant start, or field separator character in the selected ST byte.
2. 'Right digit' trigger for AB digit selection.
3. Controls for stepping the ABC.
4. Controls for determining which data (i.e., ST byte, F, or AB digit with zone) is to be used as the result byte, and controls for gating this data to the serial adder.
5. Zero detection of the selected AB digit.
6. Sign detection of the low-order digit of the selected AB byte.
7. Detection of a mark condition.
8. The 'S' trigger with associated set-reset controls.
9. Controls for setting or resetting STAT's.

The destination field is considered a pattern field and is processed one byte at a time, from left to right, under control of the STC. Each ST byte is gated to SBB for decoding and is replaced by a byte of data which, depending on decoded conditions, may be:

1. Original data of ST byte.
2. A selected digit of AB with a zone inserted into the high-order four bits.
3. A fill character, which is contained in F.

The source field is processed, one digit at a time, from left to right, under control of the ABC and 'S' trigger, which selects which digit of a byte is to be used. The selected AB digit is examined only if a digit select or significant start character appears in the selected ST byte. The selected AB digit is not necessarily used as part of the result byte, but the next digit to be processed is selected after the digit has been examined.

Microprogram Description

The flowchart for the Edit and Edit and Mark microprogram is shown in Diagram 5-411, FEMDM. At the start of the execution sequence, the fill character is gated from ST (per the STC) through the serial adder to F. A 2-cycle data-processing sequence is then started and is repeated until all destination operand bytes have been processed. Exits from this sequence are made when required for operand fetching or marking, after which this sequence is continued. The microprogram may be divided into three parts: (1) first cycle, (2) second cycle, and (3) exit conditions.

First Cycle

The first cycle is a decode cycle; no data is transferred. The selected ST byte is gated to SBB, and the selected AB byte is gated to serial adder bus A (SBA) with the digit to be examined determined by the 'right digit' trigger. The decode circuits are activated by ROS. Decoding of SBB, SBA, and the 'S' trigger governs the selection of appropriate inputs to the serial adder, and also whether the 'S' trigger is set or reset. STAT A is set if the selected source digit (in AB) is a nonzero digit. However, if a field separator character is decoded at SBB, STAT A is reset.

STAT E is set if an invalid digit is decoded in SBA(0–3). A 1 is added to D (except for the first entry from another sequence) to keep the byte address in D at the same value as the STC for use in the marking sequence. A mark condition is detected and latched for a branch condition of the Edit and Mark instruction.

Second Cycle

At the start of this cycle, data is gated to the serial adder by hardware controls as explained in the first cycle. The second cycle performs the following control functions:

1. The serial adder output is gated back to the selected ST byte, and the appropriate mark trigger is set.
2. The STC is incremented, and the LL count in E(8–15) is decremented by ROS control.
3. The ABC is incremented by hardware controls.
4. If required, the 'digit select' trigger is complemented. This action is conditional on the following:
   a. The digit selection of AB is changed only if a significant start or digit select character was decoded during the first cycle.
   b. When a sign code is decoded in SBA(4–7) at the time bits 0–3 are selected for examination, the low-order digit (sign) is skipped by stepping the ABC and leaving the 'right digit' trigger reset.
5. If required, exit to a separate routine is made via an eight-way ROS branch, for end-op, operand fetching, or marking. If no exit conditions exist, the execution sequence is repeated.

Exit Conditions

Exits from the data processing sequence are made when one or more of the following conditions exist:

1. Edit and Mark instruction is being executed and a mark condition is detected.
2. LL = 0 or STC = 7.
3. ABC = 7.

Where more than one of the above conditions exists, a branch is made to the proper sequence in the order they are listed above. An explanation of each sequence is given below:

1. Exit on Detection of Mark Condition
   Exit to the mark sequence is made regardless of other branch conditions. Special action is taken to return counter values to what they were before entering the mark sequence, so they can be retested. STAT H is set if the ABC has just stepped from 7 to 0, to record this condition. The contents of AB are destroyed by gating E(8–15) + 1 via the parallel adder to A, and the contents of T via the parallel adder to B. E(8–15) is cleared, and GPR1 is transferred to T using E(12–15) + 1 as the LAR address. The contents of D, the byte address of the last byte processed, are placed into T(40–63). T(32–39) is retained by gating it through the serial adder and back to T at the same time the D-PAL-T transfer occurs. The contents of T are now transferred back to GPR1. Registers and counters are restored to their original contents. The source operand is replaced in AB by refetching it from main storage, and a test is made, via a ROS branch, for any other exit condition which may have been present at the time the mark sequence was started. If no other exit condition exists, the data-processing sequence is resumed.
2. Exit on LL = 0 or STC = 7 (End-Op or Destination Fetch)
   STAT D is set if the ABC also equals 7, and a destination store is started and a test is made for invalid data. If STAT E has been set, an interruption code trigger is set and an end-op sequence is started. If STAT E is not set, a test is made for an end-op condition via a ROS branch. If the LL count has been stepped to all 1's, an end-op sequence is started which sets the CC, restores the instruction address to the IC, and resets STAT G. If an end-op condition does not exist, D is incremented and a fetch request is initiated for the next doubleword of

destination operand. A test is made to see whether a source fetch is also required (ABC = 0 and STAT D set). If not, the data-processing sequence is resumed.

3. Exit on ABC = 7 (Possible Source Fetch)
   A further test must be made to determine whether the last byte of AB has been processed. This is determined by testing the ABC for an all-zero count (i.e., the ABC was stepped from 7 to 0 in the previous cycle). If the ABC is not zero, the data-processing routine is restarted; otherwise, the IC is incremented by 8, and a fetch is initiated for the next doubleword of source operand. This source fetch sequence is common to all VFL logical instructions and incorporates the word-overlap test. However, this test does not affect the edit operation. The source doubleword from main storage is gated from the SDBO to AB, and the data-processing sequence is resumed.

SHIFT

Four logical shift instructions are available:

1. Shift Left Single. Shifts a 32-bit operand left.
2. Shift Left Double. Shifts a 64-bit operand left.
3. Shift Right Single. Shifts a 32-bit operand right.
4. Shift Right Double. Shifts a 64-bit operand right.

The second operand address is not used to address data. Rather, its low-order six bits indicate the number of bit positions to be shifted; the rest of the address is ignored.

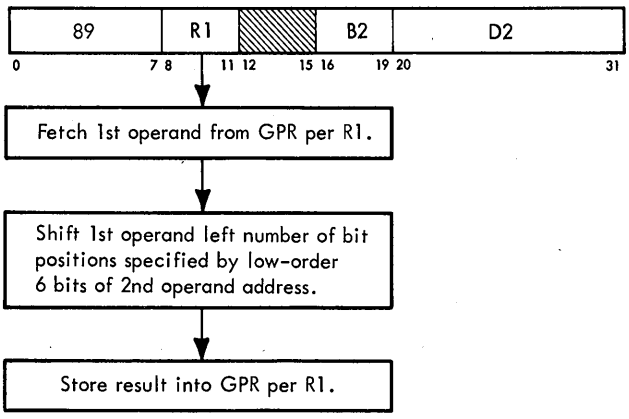
Shifting is accomplished as follows:

1. Left 1 from T to PAA.
2. Left 2 from AB to PAB.
3. Left 4 from PAA or PAB to PAL.
4. Right 4 from PAA or PAB to PAL.

Shifts of right 3 or less are obtained by combining left 1, left 2, or left 3 shifts with a right 4 shift.

Shift Left Single, SLL (89)

- Shift 1st operand (in GPR, per R1) left number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

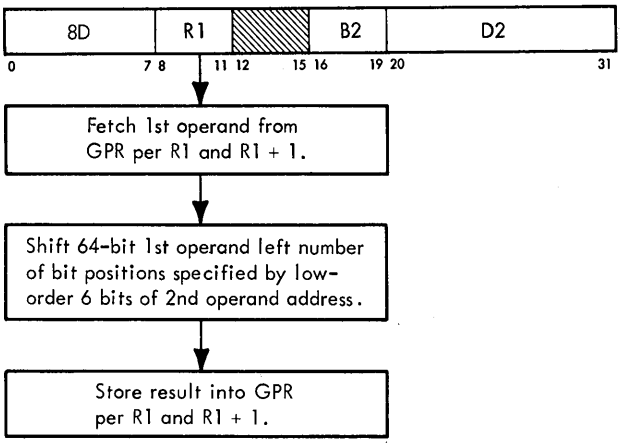


- Conditions at start of execution:
  1st operand is in S and T.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Left Single, SLL, instruction shifts the 32-bit first operand left the number of bit positions specified by the low-order six bits of the second operand address. All 32 bits of the GPR participate in the shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated low-order GPR positions. This instruction shares the same microprogram as the fixed-point Shift Left Single, SLA, instruction (Section 2 of this chapter).

Shift Left Double, SLDL (8D)

- Shift 1st operand (in GPR, per R1 and R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

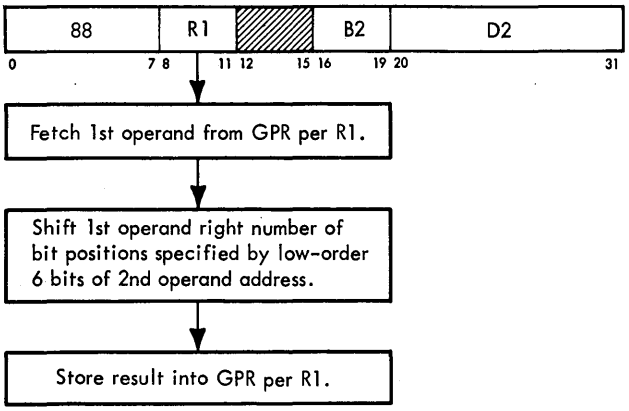


- Conditions at start of execution:
  1st operand is in ST.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Left Double, SLDL, instruction shifts the 64-bit first operand left the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction specifies an even/odd pair of GPR's and must contain an even GPR address. An odd value for R1 is a specification exception and causes a specification program interruption. All 64 bits of the even/odd GPR pair participate in the shift. High-order bits are shifted out of the even-numbered GPR without inspection and are lost. Zeros are supplied to vacated low-order positions of the odd-numbered GPR. This instruction shares the same microprogram as the fixed-point Shift Left Double, SLDA, instruction (Section 2 of this chapter).

Shift Right Single, SRL (88)

- Shift 1st operand (in GPR, per R1) right number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

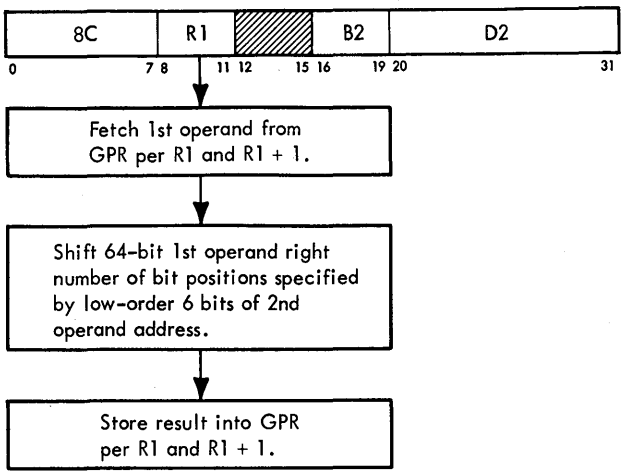


- Conditions at start of execution:
  1st operand is in S and T.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Right Single, SRL, instruction shifts the 32-bit first operand right the number of bit positions specified by the low-order six bits of the second operand address. All 32 bits of the GPR participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated high-order GPR positions. This instruction shares the same microprogram as the fixed-point Shift Right Single, SRA, instruction (Section 2 of this chapter).

Shift Right Double, SRDL (8C)

- Shift 1st operand (in GPR, per R1 and R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:



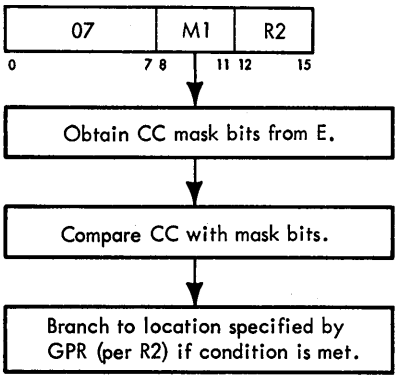
- Conditions at start of execution:
  1st operand is in ST.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Right Double, SRDL, instruction shifts the 64-bit first operand right the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction specifies an even/odd pair of GPR's and must contain an even GPR address. An odd value for R1 is a specification exception and causes a specification program interruption. All 64 bits of the even/odd GPR pair participate in the shift. Low-order bits are shifted out of the odd-numbered GPR without inspection and are lost. Zeros are supplied to vacated high-order positions of the even-numbered GPR. This instruction shares the same microprogram as the fixed-point Shift Right Double, SRDA, instruction (Section 2 of this chapter).

This section discusses the nine branching instructions. The instructions use the RR, RX, and RS formats. For a discussion of branching, operand addressing, instruction formats, data flow, and program interruptions, see Chapter 1.

BRANCH ON CONDITION, BCR (07)

- Branch to location specified by GPR (addressed by R2) if state of CC is as specified by M1.
- RR format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request for branch-to instruction has been issued per D if branch is successful.
  Instruction is in E.
- If branch is unsuccessful, 3-cycle storage request to refill Q will be issued per IC, if required.
- Branch is unsuccessful if R2 = 0 or if condition is not met.

The Branch on Condition, BCR, instruction, which has an RR format with an op code of 07, replaces the next sequential instruction address with the branch address located in the GPR specified by R2 if the CC agrees with the corresponding mask bit(s) in the M1 field. The M1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four CC's (0, 1, 2, and 3) as follows:

M1 Field	Mask Position Value	CC
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the CC has a corresponding mask bit(s) of 1.

When a branch is to be made on more than one CC, the pertinent CC's are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made on CC's 0 and 1.

When all four mask bits are 1's, that is, the mask position value is 15, the branch is unconditional. When all four mask bits are 0 or when R2 = 0, the branch instruction is equivalent to a No-Operation.

At the start of execution, the instruction is in E and the branch address is in D. For a BCR instruction, the storage request can be generated from two possible places, depending upon whether the branch is successful (Diagram 5-501, FEMDM). If the branch is successful, the storage request is generated per D. If the branch is unsuccessful, the storage request is generated per the IC if Q needs to be refilled. Because the CC's, which have to be compared with the mask bits, are set during the execution phase of a previous instruction and are tested during I-Fetch of the branch instruction, success of the branch can be determined beforehand. Therefore, the BCR instruction knows whether it is successful or unsuccessful before instruction execution.

The branch-to address is always placed in D by the I-Fetch sequence. If the branch is successful, a request is made per D by means of the 'I-Fetch reset' micro-order. The correct halfword within the doubleword from main storage is then gated into Q, and from Q to R per D(21,22). The contents of D are updated by 8 and placed into the IC to address the next sequential doubleword from main storage. If the branch is unsuccessful, the storage request is issued per the IC, if Q needs to be refilled, during I-Fetch (by means of the 'I-Fetch reset' micro-order), and the data from main storage is gated to Q during the execution of the unsuccessful branch.

Successful Branch

The 'execute' and 'PSC' triggers are reset if the branch is successful. The triggers are set if the branch instruction is the subject instruction of an Execute instruction.

The contents of D are transferred to PAA(40–63). Then 8 is added to PAA, and the result (address of next doubleword to be operated on) is transferred to the IC. D(21,22) is now tested. If D(21,22) = 11, it signifies that the next instruction to be executed, when the data is gated into Q from the SDBO, occupies the last halfword of Q, and a request must be made to obtain additional instructions for Q. If D(21,22) equals a value other than 11,

then the next instruction to be executed is in some halfword other than the last halfword of Q.

Assume that D(21,22) = 11. In this case, a storage request per the updated instruction address in the IC is given to refill Q. At this time, the data (branch-to instruction) that was requested during I-Fetch of the branch instruction is present at the SDBO and is gated into Q. From Q, the data is gated to R per D(21,22), thus placing the last halfword of Q into R.

The contents of the IC are transferred to the parallel adder, where they are updated by 8 and replaced into the IC to address the next doubleword from main storage. After the IC has been updated, the next sequential doubleword (requested during execution of the branch instruction) is gated from the SDBO into Q. A normal end-op cycle completes the operation.

Now assume that D(21,22) equals a value other than 11 on a successful branch. This condition means that the next instruction to be executed is contained in either the first, second, or third halfword of Q when the data from storage is gated into Q. The data which was requested during I-Fetch of the branch instruction is now present at the SDBO and is gated into Q. The halfword that contains the next instruction to be executed is then gated into R per D(21,22). Format decoding is normally accomplished from R and instruction address decoding from the IC. Because the data to be decoded has just been placed into R and the IC, it is not yet stable and therefore cannot be decoded during this cycle. Rather than delaying a cycle until the information is stable, a branch end-op cycle is taken. This cycle allows decoding of the halfword (containing the next instruction) from the SDBO as the data is transferred from the SDBO to Q and decoding of the instruction address from D.

Unsuccessful Branch

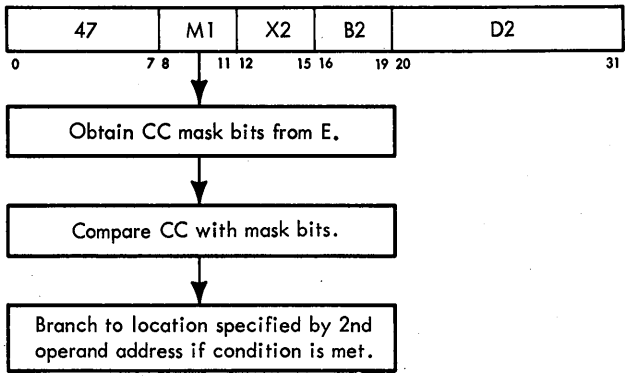
Assume, now, that the branch had been found to be unsuccessful (point B, Diagram 5-501, Sheet 2). Because the storage request was generated per the IC, IC(21,22) is now tested. Assume that IC(21,22) = 11. This value means that the next instruction to be executed is located in the last halfword of the doubleword from main storage that contains the branch instruction. This instruction is in R. Because a request to refill Q per the IC was made during I-Fetch, the IC must be updated. Accordingly, the contents of the IC are now transferred to PAB(40–63). Then 8 is added to PAB, and the result is transferred to the IC and D. At this time, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and is gated into Q. The 'execute' trigger is now tested. If the trigger is set, the IC is reduced by 8 (because 8 had been added to it and the I-Fetch request was blocked by STAT G being set by the Execute instruction), a normal end-op cycle is taken, and an address-store-compare refill of Q will be performed. If the 'execute' trigger is reset, a branch end-op cycle is taken.

Now assume that IC(21,22) = 00, 01, or 10 and that the branch instruction was unsuccessful (point C, Diagram 5-501, Sheet 2). In this case, the next instruction to be executed is in either the first, second, or third halfword of Q, and Q does not need to be refilled. Therefore, a normal end-op cycle is taken, the next instruction format is decoded from R, and the instruction address is decoded from the IC.

A unique situation occurs for the BCR instruction when the 'PSC' trigger is set by an Execute instruction. This situation causes a Q-register refill following the Execute instruction. A branch is taken to the address-store-compare ROS microprogram, 8 or 16 is subtracted from the IC to select the doubleword that contains the instruction following the Execute instruction, and a storage request per the IC is made for that doubleword.

BRANCH ON CONDITION, BC (47)

- Branch to location specified by 2nd operand address if state of CC is as specified by M1.
- RX format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request for branch-to instruction has been issued per D, if branch is successful.
  First 16 bits of instruction are in E.
- If branch is unsuccessful, 3-cycle storage request to refill Q will be issued per IC, if required.

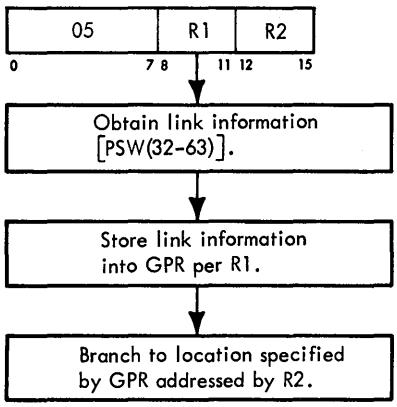
The BC instruction is similar to the BCR instruction, differing only when IC(21,22) = 00 when the branch is unsuccessful (point B, Diagram 5-501, Sheet 2). This value means that the next instruction to be executed is to come from the first halfword of the doubleword which has been requested during I-Fetch. The contents of the IC are updated by 8 and placed into the IC and D. The doubleword from main storage requested during I-Fetch is

now gated from the SDBO to Q. The first halfword from Q is then transferred to R.

The 'execute' trigger is now tested. If reset, the branch instruction ends in a branch end op, and format decoding of the next instruction takes place off the SDBO because R is not stable. If the 'execute' trigger is set, the branch instruction was the subject instruction of an Execute instruction, and the present contents of the IC are incorrect because the IC was increased by 8 during this instruction. The contents of the IC are thus reduced by 8 and replaced in the IC. A normal end op completes the operation. Because the BC instruction was the subject of an Execute instruction, the return to the proper next instruction occurs during I-Fetch, as discussed in the BCR instruction.

BRANCH AND LINK, BALR (05)

- Store PSW (32–63), link information, into GPR (addressed by R1) and branch to location specified by GPR (addressed by R2).
- RR format:



- Conditions at start of execution:
  2nd operand is in A, B, and D.
  3-cycle storage request has been issued per D for branch-to instruction.
  Instruction is in E.
- Link information consists of:
  Instruction length code.
  CC.
  Program mask bits.
  Address of next sequential instruction (link address).
- Link information is stored whether branch is successful or unsuccessful.
- If 'execute' trigger is set, link address is address of instruction following Execute instruction.
- Branch is unsuccessful if R2 = 0.

The Branch and Link, BALR, instruction, which has an RR format with an op code of 05, stores the address of the next sequential instruction. Stored with the address is link information containing the instruction length code, the CC, and the program mask bits. The instruction length code stored will be either 1 or 2. If the instruction length code stored is 2, the BALR instruction is the subject of an Execute instruction. If during a BALR operation the R2 field is equal to zero, the branch is considered unsuccessful.

The purpose of the BALR instruction is to branch to a subroutine and provide a means of returning from the subroutine to the main flow of instructions in a program. How this is accomplished is shown in Figure 3-26. When processing the main instruction flow and a BALR instruction is encountered, the address of the instruction which sequentially follows the BALR in the main instruction flow is stored in LS. For the example illustrated in Figure 3-26, the address of the next sequential instruction is 14 and is stored in GPR9. (If the BALR instruction was in address 14 of the main program, then the address stored would be 16.) Once the instruction address is stored, the branch to the subroutine occurs. The subroutine is performed and, when completed, a branch instruction could be issued using the address that was stored during the BALR instruction as the branch address to return to the main flow of instructions. After returning to the main flow of instructions, the program will continue in its normal manner, processing the remaining instructions.

In determining the address which is to be stored as the link address, IC(21,22) and the 'execute' trigger must be tested (Diagram 5-502, Sheet 2, FEMDM). If IC(21,22) = 11 and the 'execute' trigger is set (indicating the BALR is the subject of the Execute instruction and the Execute instruction is located in the second and third halfwords of its doubleword), 16 is subtracted from the IC and placed into T. Thus, if it is necessary to return to the main flow of instructions, the instruction which will be performed next is that instruction sequentially following the Execute instruction. If IC(21,22) does not equal 11 or the 'execute' trigger is reset, 8 is subtracted from the IC and the value is placed into T.

Unsuccessful Branch

E(12–15), which contains the address of the GPR which has the branch address, is examined. If E(12–15) = 0, branching is not to take place and a No-Operation occurs. If E(12–15) equals anything other than zero, branching occurs unconditionally. First assume that the branch is unsuccessful [E(12–15) equals zero]. IC(21,22) is tested. If IC(21,22) = 11, it indicates that the next instruction to be executed is in R (this instruction is the last halfword of the doubleword in Q) and a new doubleword must be placed into Q. If IC(21,22) equals any other value, the next

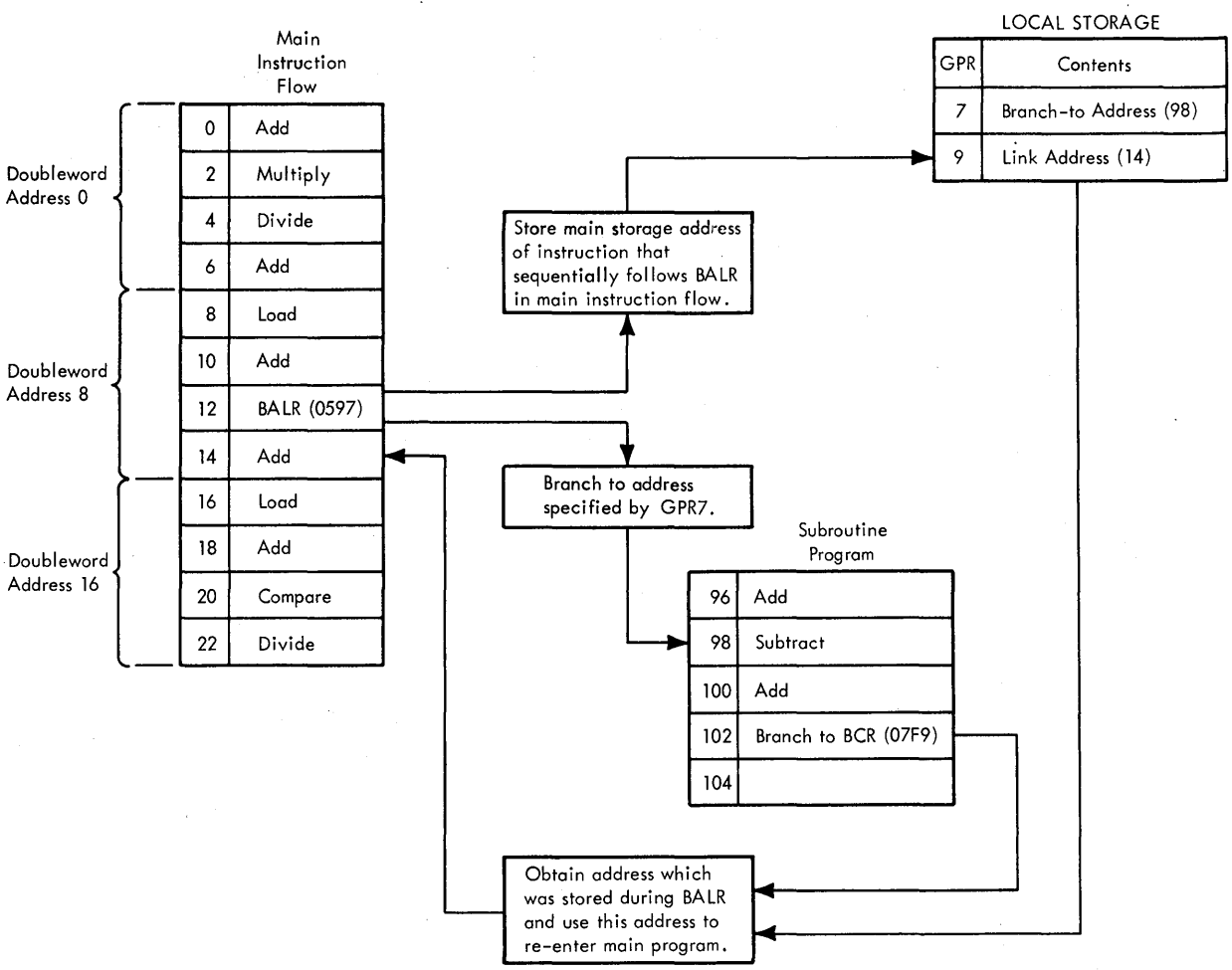


Figure 3-26. Example of Use of Branch and Link Instruction

sequential instruction is also located in R but Q contains data which is still correct and may be operated on. In either case, the remainder of the link data [PSW(32–39)] is placed into T(32–39) and from there transferred to the GPR per E(8–11). If IC(21,22) equals a value other than 11, a normal end op is taken after storing the data and the next instruction is decoded from R. If IC(21,22) = 11, after storing the link information, a 3-cycle storage request is issued per the IC to obtain the next sequential doubleword to refill Q. The 'execute' trigger is again tested. If set, a normal end op is immediately taken and the next instruction to be executed is the instruction which sequentially follows the Execute instruction (of which the BALR instruction was the subject instruction). (This action is accomplished during the next I-Fetch by means of the ASC micro-program branch which will refetch the instruction following the Execute instruction.) If the 'execute' trigger is reset, the next instruction to be executed is in R. The IC is updated by 8 to select the next sequential doubleword after the one just requested. IC(21,22) is again tested; if it equals 11, the data on the SDBO is gated to Q and a branch end-op cycle is taken.

Successful Branch

Now assume that E(12–15) does not equal zero (indicating a successful branch). The 'PSC' and 'execute' triggers are reset by a combination of the 'T6' and 'M4' micro-orders. PSW(32–39) is transferred to T(32–39). The doubleword containing the branch-to instruction (requested during I-Fetch of the branch instruction) is now present at the SDBO and is gated to Q.

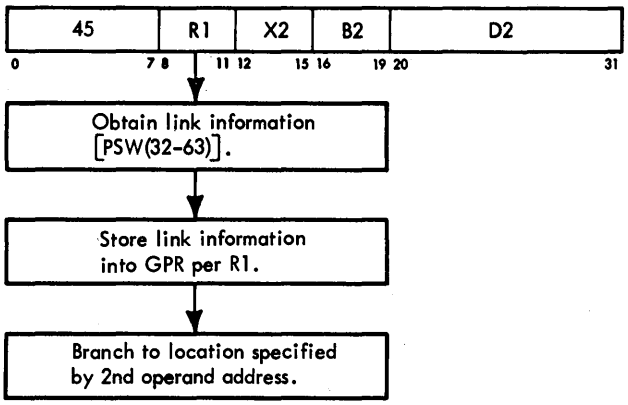
From Q, the correct halfword is transferred to R per D(21,22). Then 8 is added to D, and the result is placed into the IC to address the next sequential doubleword from main storage. The data in T is now transferred to the GPR per E(8–11).

D(21,22) is tested for a value of 11. If it does not equal 11, the data in R and the IC is not yet stable, thus preventing decoding of the next instruction from R or

decoding of the instruction address from the IC. A branch end-op cycle is taken, and the next instruction is decoded off the SDBO which, at this time, is stable. If D(21,22) = 11, the next instruction is located in the last halfword of the doubleword requested during I-Fetch. From Q, the last halfword is transferred to R per D(21,22). Because the halfword that contains the next instruction to be executed is the last halfword of the doubleword, Q must be filled with a new doubleword to allow continuous operation. Then 8 is added to D, and the result is transferred to the IC. A 3-cycle storage request is issued per the IC to obtain the next doubleword. The address of the doubleword is tested for validity; if the address is invalid, the 'I-Fetch invalid address' trigger is set. The IC is updated by 8. By this time, the requested doubleword is present at the SDBO and is gated to Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded off R.

BRANCH AND LINK, BAL (45)

- Store PSW(32–63), link information, into GPR (addressed by R1) and branch to location specified by 2nd operand address.
- RX format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  First 16 bits of instruction are in E.
- Link information consists of:
  Instruction length code.
  CC.
  Program mask bits.
  Address of next sequential instruction (link address).
- Link information is stored whether branch is successful or unsuccessful.
- BAL is unconditional branch.
- If 'execute' trigger is reset and ABC = 0, IC reflects correct address and is stored as link address.
- If 'execute' trigger is reset and ABC does not equal 0, IC is reduced by 8 and then stored as link address.
- If 'execute' trigger is set, link address is address of instruction following Execute instruction.

The Branch and Link, BAL, instruction stores the address of the instruction which, if the branch were unsuccessful, would be the next sequential instruction address. Stored with the address is link information consisting of the instruction length code, the CC, and the program mask bits. The instruction length code stored is 2. The BAL instruction is an unconditional branch with an RX format and an op code of 45.

At the start of execution, the first 16 bits of the instruction are in E, the branch address is in D, and a 3-cycle storage request has been generated per D for the branch-to instruction (Diagram 5-503, FEMDM). At the beginning of the operation, the last three bits of the IC are transferred to the ABC. This value will be used to determine the correct value of the IC before it is stored into LS as link address information. The contents of the IC are transferred to the parallel adder and reduced by 8. This value is then transferred to T, from where it and the remainder of the link data [PSW(32–39)] will be transferred to LS. The 'execute' trigger is then tested; if it is set, the branch operation is the subject instruction of an Execute instruction.

First assume the 'execute' trigger is reset. The ABC is now checked for all zeros. If equal to zero, it indicates that the branch instruction now being executed was located in the third and fourth halfwords of Q. Normally, when an instruction with an RX format occupies the last two halfwords of Q, a storage request is generated during I-Fetch per the IC and the IC is updated by 8. Because this is a branch instruction, however, the storage request from the IC is prevented and the IC is not increased by 8. Therefore, the address presently in T (after being reduced by 8) is incorrect and 8 must be added to it before it is stored into LS.

Assume that the ABC did not equal zero. PSW(32–39) is transferred to T(32–39). Because the ABC was not equal to zero, the link address is correct and can be stored into the GPR per E(8–11). The contents of D are then transferred to the parallel adder, increased by 8, and transferred to the IC. The IC now contains the address of the doubleword in main storage which follows the doubleword containing the branch-to instruction. At this time, a storage request for the next doubleword is issued if D(21,22) = 11. The data requested per D during I-Fetch is at the SDBO and is gated to Q. From Q, the correct halfword is transferred to R per D(21,22).

D(21,22) is now checked for a value of 11. If it is equal to 11, the next instruction to be executed is in the last

halfword of Q. The IC is then updated by 8 to address the next doubleword in main storage. By this time, the doubleword that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21,22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Now assume that the 'execute' trigger is set, indicating that the branch instruction is the subject instruction of an Execute instruction. IC(21,22) is tested for a value of 11. If it equals 11, the Execute instruction was located in the second and third halfword of its doubleword and, when in Q, I-Fetch issued a storage request and increased the IC by 8. This increase results in an address in the IC that is 16 bytes higher than the doubleword address containing the Execute instruction. Because the address that is stored as link information is the address of the doubleword containing the instruction immediately following the Execute instruction, the address has to be reduced by 16. The address in T, however, has already been reduced by 8; therefore, only 8 must be subtracted from it. The remainder of the link information is now transferred from PSW(32–39) to T(32–39).

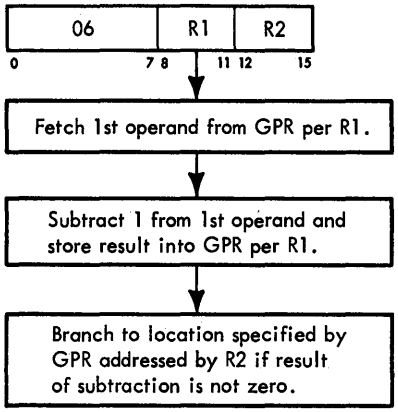
The contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC to address the next doubleword. The 'execute' trigger is now reset. A storage request for that doubleword whose address was just placed into the IC is issued if D(21,22) = 11. At this time, the data requested during I-Fetch of the branch instruction is present on the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21,22). The link address located in T is transferred to the parallel adder, where it is decreased by 8. This value is now equal to the address of the doubleword that contains the Execute instruction and is transferred to T. From T, the link information is transferred to the GPR per E(8–11).

D(21,22) is now checked for a value of 11. If it equals 11, the next instruction to be executed is in the last halfword of Q. The IC is then updated by 8 to address the next doubleword in main storage. By this time, the doubleword that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21,22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Now assume that IC(21,22) did not equal 11. This condition indicates that the Execute instruction was not in the second halfword and a storage request was not automatically generated. The link information presently in T is therefore correct and can be stored into the GPR per E(8–11). The contents of D are increased by 8 and placed into the IC. Again a storage request is generated if D(21,22) = 11. At this time, the doubleword containing the branch-to instruction is located in the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21,22). D(21,22) is tested for a value of 11, and operations continue as previously described, ending with a branch end op or a normal end op.

BRANCH ON COUNT, BCTR (06)

- Subtract 1 from 1st operand (in GPR, per R1) and, if result is not 0, branch to address specified by GPR (addressed by R2).
- RR format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  First operand is in S and T.
  Instruction is in E.

The Branch on Count, BCTR, instruction subtracts 1 from the first operand (contents of the GPR specified by R1) and, if the result does not equal zero or R2 does not equal zero, branches to the address specified by the contents of the GPR designated by R2. The result of the subtraction is stored into the first operand location. If the result of the subtraction equals zero, the next sequential instruction is executed. If E(12–15) = 0, the branch is automatically unsuccessful. The BCTR instruction has an RR format with an op code of 06.

At the start of execution, the instruction is in E, the first operand is in S and T, the branch address is in D, and a 3-cycle storage request has been issued per D for the branch-to instruction (Diagram 5-504, FEMDM). The first operand is transferred from T to B and from B to the parallel adder, where 1 is subtracted from the operand to determine whether the branch is successful. Before subtracting 1, E(12–15) is tested for zeros. As previously stated, if E(12–15) = 0, the branch is unsuccessful. Assume that E(12–15) does not equal zero. The contents of B are

transferred to the parallel adder, where 1 is subtracted from the operand; the result is transferred via T into LS. The result of the subtraction is tested for all zeros; if zero, the branch is unsuccessful; if not zero, the branch is successful.

Successful Branch

First assume that the branch is successful. The 'PSC' and 'execute' triggers are reset. Because D(21,22) indicates in which halfword the branch-to instruction is located, it is examined. If D(21,22) = 11, the instruction is located in the last halfword of the doubleword requested during I-Fetch of the branch instruction. The contents of D, therefore, are updated by 8 and transferred to the IC. By this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q. The last halfword of Q is then transferred to R per D(21,22). A 3-cycle storage request for the next doubleword is now issued per the IC. The contents of the IC are then transferred to the parallel adder, updated by 8, and transferred back to the IC to select the next doubleword from main storage. At this time, the doubleword which sequentially follows the doubleword containing the branch-to instruction is present at the SDBO and is gated into Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded from R.

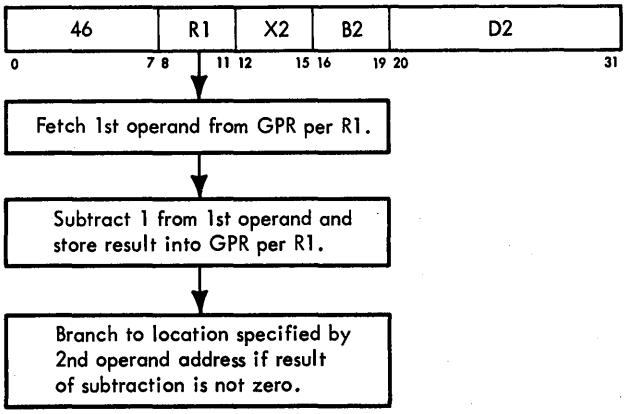
If D(21,22) did not equal 11, the branch-to instruction is located in some halfword other than the last. In this case, the contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC. At this time, the doubleword containing the branch-to instruction is present at the SDBO and can be gated into Q. From Q, the correct halfword is transferred to R per D(21,22). A branch end-op cycle is taken, and the next instruction is decoded off the SDBO.

Unsuccessful Branch

Now assume that the branch is unsuccessful. If IC(21,22) = 11 or 00, a storage request per the IC must be given during the branch execution phase to obtain the next sequential doubleword from main storage (because this action was inhibited during the end-op cycle by the branch decoder). Once the storage request is issued, the 'execute' trigger is tested. If set, it indicates that the branch instruction is the subject instruction of an Execute instruction. Therefore, a normal end-op cycle is taken to complete the operation. The data requested in this case is not gated into Q. If the 'execute' trigger is reset, IC(21,22) is tested to see whether it contains 11. If the value is 11, the IC is updated by 8 (to select another doubleword) and placed into the IC and D. By this time, the data requested by the storage request given during the execution phase of the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken, and the next instruction to be executed is decoded off R. If IC(21,22) does not equal 11, then it equals 00, and the next instruction is located in the first halfword of the doubleword requested during the execution phase of the branch instruction. The IC is updated by 8, and the result is placed into D and the IC. At this time, the data requested during the execution phase is present at the SDBO. This doubleword is gated to Q; the correct halfword from Q(0–15) is transferred to R. Because the format is normally decoded off R and the data just placed in R is not yet stable, a branch end-op cycle is taken. This cycle allows the next instruction to be decoded off the SDBO, which at this time is stable.

BRANCH ON COUNT, BCT (46)

- Subtract 1 from 1st operand (in GPR, per R1) and, if result is not 0, branch to location specified by 2nd operand address.
- RX format:



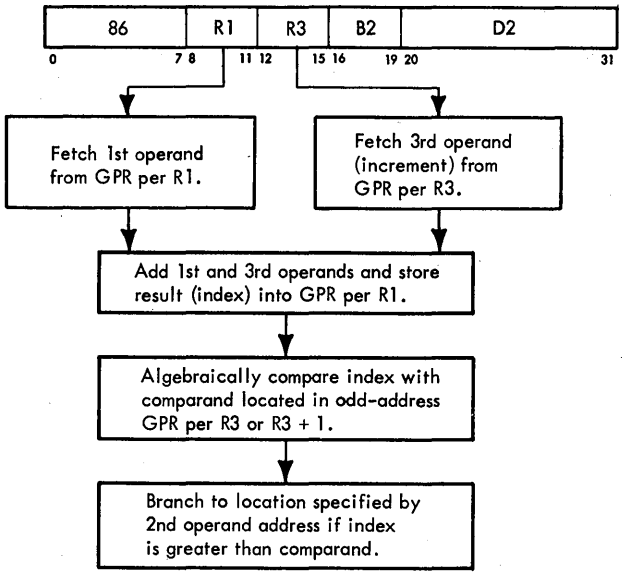
- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.

The Branch On Count, BCT, instruction is similar to the BCTR instruction except that E(12–15) is not tested for zero. Refer to Diagram 5-504 for a flowchart of the BCT instruction.

BRANCH ON INDEX HIGH, BXH (86)

- Add increment (3rd operand; in GPR, per R3) to 1st operand (in GPR, per R1), algebraically compare result (index) with comparand (in odd-address GPR specified by R3 or R3 + 1), and, if index is greater than comparand, branch to location specified by 2nd operand address.

- RS format:



- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.
- Sum of 1st and 3rd operands is always stored whether branch is successful or not.
- Comparand address (R3 or R3 + 1) must be odd.

The Branch on Index High, BXH, instruction, which has an RS format with an op code of 86:

1. Adds the third operand to the first operand.
2. Stores the result (index) into the GPR addressed by R1.
3. Compares the index with a comparand obtained from a GPR addressed by R3 or R3 + 1.
4. Branches if the sum is greater than the comparand.

At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, the branch address is in D, and a 3-cycle storage request has been issued per D for the branch-to instruction (Diagram 5-505, FEMDM). To allow the third operand to be placed into T without destroying the first operand, the first operand is transferred to B. The third operand is then transferred from the GPR per E(12–15) and placed into T. The two operands are added, and the result is transferred to B. The comparand, the value that the sum of the two operands (index) is compared with, is now transferred from LS and placed into T. The contents of B and the 2's-complement of T are transferred to the parallel adder and added to determine whether the branch is successful. IC(21,22) is tested for either 11 or 00. If the branch is unsuccessful, this test sets up conditions to issue a storage request per the IC to obtain the next sequential doubleword after the doubleword containing the branch instruction.

Assume that IC(21,22) = 11 or 00. D(21,22) is now tested to determine where in the doubleword the next instruction is located. If D(21,22) = 11, the next instruction to be executed is contained in the last halfword of the doubleword requested during I-Fetch of the branch instruction. The PAL's and E(7) are now tested (by means of the 'J47 ≠ 0' micro-order) to determine whether the branch is successful. First assume the branch is successful; that is, the PAL's are positive (index greater than the comparand) and E(7) = 0. The contents of B (index) are transferred to T, and from there to the GPR per E(8–11). The 'PSC' and 'execute' triggers are reset by means of the 'TIF' micro-order. At this time, the doubleword requested during I-Fetch is present at the SDBO and is gated into Q. The halfword containing the next instruction to be executed is then transferred to R per D(21,22). A test is made to re-establish that D(21,22) = 11. Assuming the original conditions still exist, the contents of D are now updated by 8 and placed into the IC. This action allows the selection of the next doubleword in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next doubleword from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

Now assume the branch is unsuccessful. That is, the PAL's are not positive and E(7) = 0. The contents of B (index) are transferred to T, and from there to the GPR per E(8–11). Because the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R. If IC(21,22) = 11 or 00, a 3-cycle storage request is issued per the IC, which at this time contains the address of the doubleword that sequentially follows the doubleword containing the branch instruction. The 'execute' trigger is now tested. If set, it indicates that the branch instruction

was the subject instruction of an Execute instruction and a normal end-op cycle is taken, completing the operation. If the 'execute' trigger is reset, IC(21,22) is tested for a value of 11. Recall that, when IC(21,22) was previously tested, it was checked for a value of either 11 or 00. To proceed sequentially in the program without any delay, it is now necessary to determine which value the IC contains. First, assume that IC(21,22) = 11. This value indicates that the next instruction occupies the last halfword of the doubleword in which the branch instruction is located and is presently in R. Recall that R is where format decoding of an instruction occurs. This situation being the case, the IC is updated by 8 to address the doubleword that is 16 bytes from the doubleword address containing the branch instruction. The data that was requested when it was found that the branch was unsuccessful is now present at the SDBO (Diagram 5-505, Sheet 3), and can be gated into Q. A branch end-op cycle is taken, completing the operation. The next instruction is decoded off the SDBO when the data is transferred to Q.

Now assume that IC(21,22) = 00. In this case, the data to be executed is located in the first halfword of the doubleword requested during the execution of the branch when the branch instruction was found to be unsuccessful. The IC is updated by 8. At this time, the data is present at the SDBO and can be gated into Q. The first halfword in Q is transferred to R. Recall that the format for the next instruction is normally decoded off R. Because the next instruction to be executed has just been transferred into R, this data is not yet stable and cannot be decoded. A branch end-op cycle is therefore taken. This cycle allows the next instruction format to be decoded off the SDBO. The SDBO is stable at this time and therefore can be used.

Now assume that when D(21,22) was tested for 11 some other value was found. Again, conditions are tested to see whether the branch is successful (Diagram 5-505, Sheet 2). If successful, the data in B is transferred to T and from there to the GPR per E(8–11). Also, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and can now be gated into Q. From Q, the halfword containing the next instruction is transferred to R per D(21,22). If D(21,22) = 11, the contents of D are now updated by 8 and placed into the IC. This action allows selection of the next doubleword in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next doubleword from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

If D(21,22) = 11 and the branch is unsuccessful, the contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). If IC(21,22) = 11 or 00, a 3-cycle storage request is issued per the IC. [If IC(21,22) equals a value other than 11 or 00, a normal end cycle is taken.] This request is for the doubleword that sequentially follows the doubleword in main storage containing the branch instruction. The 'execute' trigger is tested next. From this point on, the operation is the same as that previously discussed for an unsuccessful branch.

Assume, now, that when IC(21,22) was initially tested for either 11 or 00, neither of these values was present (Diagram 5-505, Sheet 2). Again D(21,22), the PAL's, and E(7) are tested to determine whether the operation is a successful branch and where the next instruction is located in the doubleword address being branched to. Assume that D(21,22) = 11 and the branch is successful (Diagram 5-505, Sheet 3). Operation from this point on is identical with a successful branch, as previously described, when D(21,22) = 11.

Now assume that D(21,22) = 11 and that the branch is unsuccessful. The contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). Because the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R.

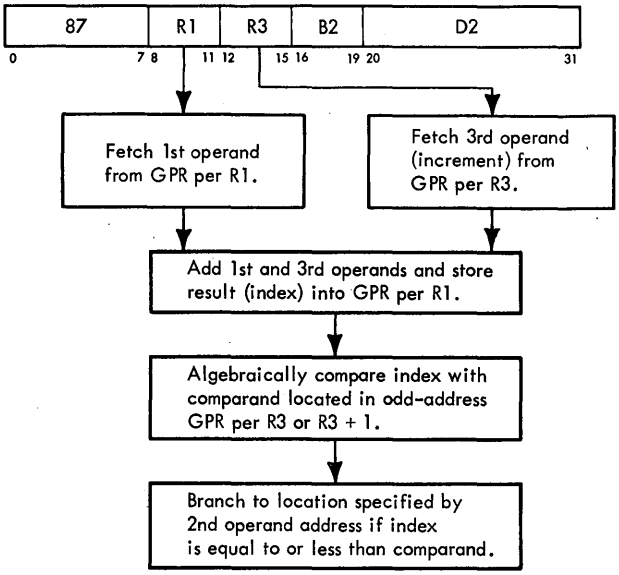
Now assume that D(21,22) did not equal 11 and the branch is successful (Diagram 5-505, Sheet 3). The contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). At this time, the data requested during I-Fetch of the branch instruction is present at the SDBO and is gated to Q. From Q, the correct halfword containing the branch-to instruction is gated into R per D(21,22). The contents of D are then updated by 8 and transferred to the IC to select the next doubleword from main storage. Because the instruction to be executed has just been placed into R and is not yet stable, a branch end-op cycle is taken and the instruction format is decoded off the SDBO.

If the branch is unsuccessful and D(21,22) does not equal 11, the operation is identical with the case where D(21,22) = 11 and the branch is unsuccessful. A normal end-op cycle is taken.

BRANCH ON INDEX LOW OR EQUAL, BXLE (87)

- Add increment (3rd operand; in GPR, per R3) to 1st operand (in GPR, per R1), algebraically compare result (index) with comparand (in odd-address GPR specified by R3 or R3 + 1), and, if index is equal to or is less than comparand, branch to location specified by 2nd operand address.

- RS format:

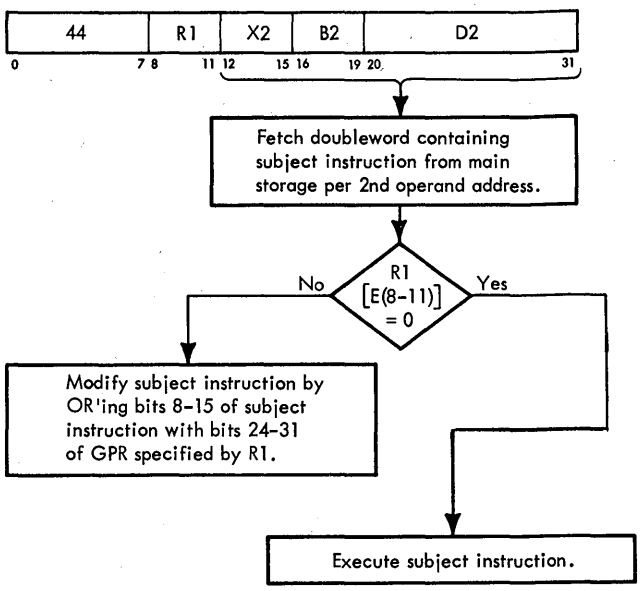


- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.
- Sum of 1st and 3rd operands is always stored whether branch is successful or not.
- Comparand address (R3 or R3 + 1) must be odd.

The Branch on Index Low or Equal, BXLE, instruction is similar to the BXH instruction except that branching occurs on a low or equal result (Diagram 5-505).

EXECUTE, EX (44)

- Execute subject instruction at location specified by 2nd operand address. Subject instruction may be modified by 1st operand (in GPR, per R1) if E(8–11) is not equal to zero.
- RX format: (See adjoining column.)
- Conditions at start of execution:
  Address of subject instruction is in D.
  3-cycle storage request for subject instruction has been issued per D.
  1st operand is in S and T.
  First 16 bits of instruction are in E.



- Modification of subject instruction is accomplished by OR'ing bits 8–15 of subject instruction with bits 24–31 of 1st operand.
- If subject instruction is an Execute instruction, a program execute interruption occurs.
- If effective address of Execute instruction is odd, a program specification interruption occurs.
- 'Execute' trigger is set to indicate next instruction is subject of Execute instruction.
- 'PSC' trigger is set to indicate that Q data is not valid and needs to be refilled.
- If program interruptions are pending, normal end-op cycle is taken; if not, branch end-op cycle is taken.

The Execute, EX, instruction, which has an RX format with an op code of 44, executes a designated instruction whose address in main storage is the second operand address. This designated instruction is referred to as a subject instruction and can be modified by the contents of the first operand located in the GPR register specified by R1. Modification of the subject instruction is accomplished by OR'ing bits 8–15 of the subject instruction with bits 24–31 of the first operand. If R1 = 0, no modification takes place. The subject instruction may be 16, 32, or 48 bits long. If the subject instruction is another Execute instruction, a program execute interruption occurs and operation is suppressed. If the effective address of the EX is odd, a program specification interruption occurs.

At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, the address of the subject instruction is in D, and a 3-cycle storage request for the subject instruction has been issued per D (Diagram 5-506, FEMDM). At the beginning of execution, a test for program specification and execute

interruptions is made. If the program specification interruption is present (effective address of Execute instruction is odd), a program interruption occurs and the operation is suppressed. If a program execute interruption is present (the Execute instruction is the subject instruction of an Execute instruction), a program execute interruption occurs and the operation is suppressed. If no interruptions are present, the operation continues. The STC is loaded to 111, allowing the transfer of T(56–63) to the serial adder for modification of the subject instruction if modification is to be accomplished. The contents of D are now transferred to the parallel adder and updated by 8 to address the doubleword that follows the doubleword containing the subject instruction of the Execute instruction. This value is then transferred to D. PAL(61–63) is now transferred to the ABC to select the correct byte for modification of the subject instruction.

D(21,22) is tested to determine whether the subject instruction is contained in the last halfword of the doubleword that was requested during I-Fetch, or in some halfword other than the last. If D(21,22) = 11, the subject instruction is in the last halfword; if any other value, the subject instruction is in some other halfword. First assume that D(21,22) = 11. Because the subject instruction is located in the last halfword of the doubleword addressed during I-Fetch, there is a possibility that part of the instruction is contained in the next doubleword to be addressed. This possibility exists if the subject instruction has a format other than RR. Therefore, the next few operations determine the format of the subject instruction. By doing these tests, an extra request can be prevented, one that can cause an invalid address or protection check if the instruction has an RR format.

At this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q and AB. From Q, the last halfword is transferred to R. The sixth byte in AB is then transferred to the serial adder per the ABC. Minus 64 (1100 0000) is sent to the serial adder, where it is logically AND'ed with the op code of the subject instruction. If the op code denotes an RR format, SAL should now equal zero. 1 is then added to the ABC to transfer the last halfword of AB to the serial adder if the instruction is to be modified.

E(8–11) is now tested to see whether the subject instruction is to be modified. If E(8–11) = 0000, the subject instruction is not to be modified; if any other value, the instruction is to be modified. First assume that E(8–11) = 0000. SAL(0–7) is now tested. Recall that the value in SAL indicates whether the subject instruction has an RR format, and recall that it has already been determined that the subject instruction is contained in the last halfword of Q. Therefore, assume that SAL(0–7) = 0. Because this value indicates that the subject instruction is an RR instruction, there is no need to issue a storage request for the next instruction because there is no information in that doubleword that will affect the operation of the RR instruction. The request for the next doubleword occurs during I-Fetch of the RR instruction.

The last byte in AB is then transferred to T via the serial adder per the ABC and STC. From T, the data is transferred to R. The 'PSC' and 'execute' triggers are set. A test is made for a pending program interruption. If one exists, a program interruption cycle is taken; if there is no interruption, the contents of R are transferred to E and STAT G is set. The setting of STAT G prevents the occurrence of interruptions and the premature pre-fetching of the next instruction from interfering with the execution of the subject instruction. A branch end-op cycle completes the operation.

Now assume that SAL(0–7) does not equal zero, indicating that the subject instruction has some format other than RR. So that the complete word may be decoded before instruction execution, the doubleword that sequentially follows the subject instruction must be requested. The last byte in AB is transferred to T via the serial adder per the ABC and STC. The 'PSC' trigger is set (insuring return to the main instruction flow upon execution of the subject instruction). This action causes an ASC exceptional condition branch to occur during the I-Fetch following execution of the subject instruction. The ROS microprogram subtracts the correct amount from the IC to select the doubleword containing the instruction following the Execute instruction, and issues a request for that doubleword. A 3-cycle storage request is issued for the next sequential doubleword. The microprogram waits (two storage cycles) until the data requested is present at the SDBO, at which time the data is gated to Q. The 'execute' trigger is set, and a test is made for a pending interruption. If an interruption is present, a program interruption cycle is taken; if not, the data in R is transferred to E and STAT G is set. A branch end-op cycle completes the operation.

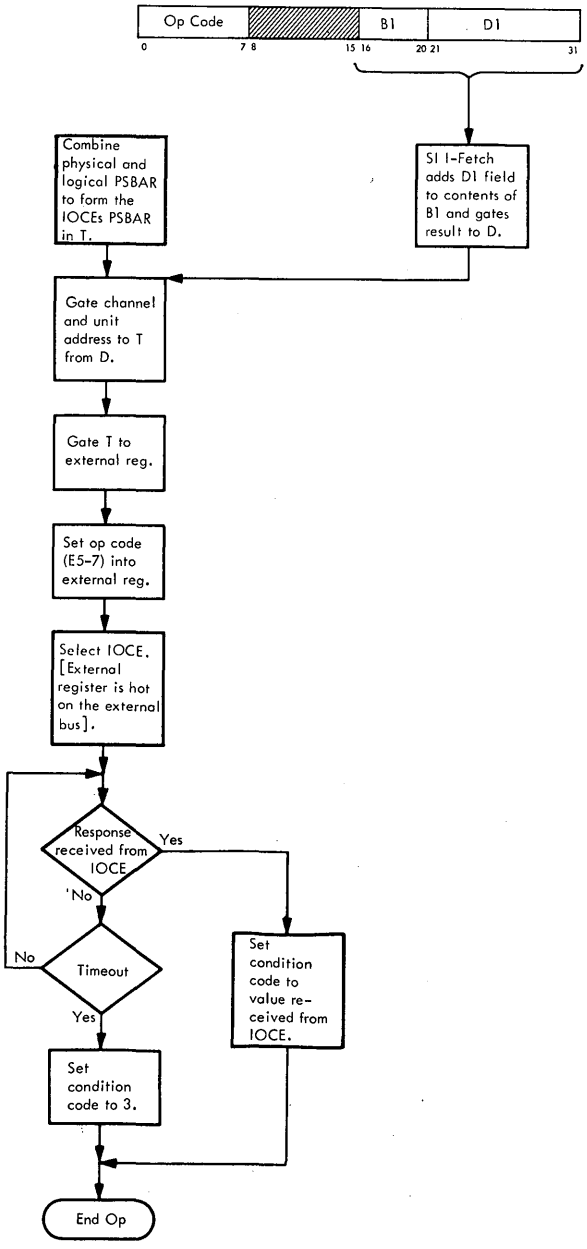
If when E(8–11) is tested some value other than 0000 is found, the subject instruction is to be modified. SAL(0–7) is tested to determine whether the subject instruction has an RR format. Assume an RR format. The last byte in AB is then transferred to the serial adder per the ABC. This byte is then OR'ed with the last byte of ST which was transferred to the serial adder per the STC. The results of this OR'ing are then transferred to ST per the STC. The data in T is transferred to R. The 'execute' trigger is set, and the operation continues in the same manner described previously.

Now assume that SAL(0–7) contains some value other than zero. In this case, after the subject instruction is modified, a storage request for the next doubleword must be made. The microprogram waits (two storage cycles) until the data is present at the SDBO, after which the data is transferred to Q. From this point on, the operation is identical with that described for an RR instruction.

Now return to the point where D(21,22) is tested to see whether the subject instruction occupies the last halfword of the doubleword requested during I-Fetch. If D(21,22) equals some value other than 11, a storage request is not issued during execution of the Execute instruction. E(8–11) is now tested to determine whether the subject instruction is to be modified. If E(8–11) does not equal 0000, the instruction must be modified. After the data is placed into Q and AB, and the correct data is placed into the serial adder per the ABC, operation is identical with that of an RR instruction that is to be modified.

If E(8–11) = 0000, however, the subject instruction is not to be modified. Therefore, the correct halfword in Q need only be transferred to R per D(21,22). The 'PSC' and 'execute' triggers are set, and the test for a pending interruption is made. The operation continues in the same manner described previously.

- Five I/O instructions: Start I/O, Test I/O, Halt I/O, Test Channel, and Set Program Controlled Interrupt.
- All I/O instructions have SI format and share a common ROS microprogram in CE.
- SI format:

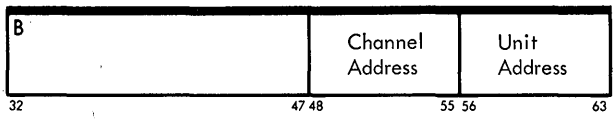


- Conditions at start of execution:
  First 15 bits of instruction in E.
  First operand is not applicable.
  Operand address (address of channel and I/O unit) in D.
- Condition codes specify status of IOCE and I/O unit.
- All instructions use SI format.
- For a detailed discussion read the following text and refer to Diagram 5-701 FEMDM.

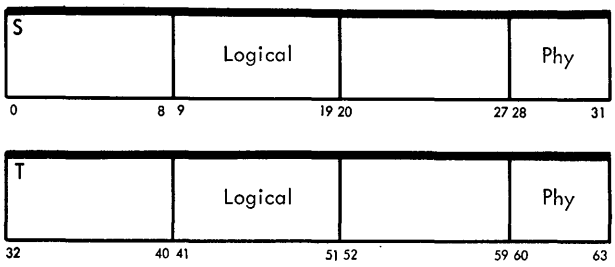
The CE issues I/O instructions to initiate I/O operations, which are under control of the IOCE. Since the I/O initialization sequence is the same, the CE uses the same ROS microprogram for all I/O instructions. This microprogram assembles all information that an IOCE needs and gates it to the proper IOCE. Table 3-13 lists the I/O instructions and the resulting condition codes.

An I/O instruction may be executed only when the CE is in the Supervisor state. Thus, the first step in the microprogram, is to determine the state of the CE. If the CE is not in the Supervisor state, a privileged-operation check occurs and causes a program interruption.

If the CE is in the supervisor state, execution of an I/O instruction begins by gating the channel and unit address from D to B.



The logical and physical PSBARs are gated to S and T.

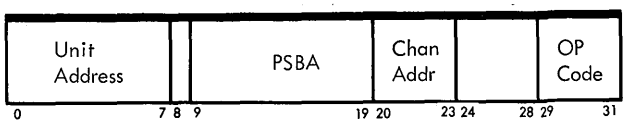


This data must next be combined into a one word format so that it can be gated over the external bus to the proper IOCE.

The IOCE must be sent the issuing CE's preferential storage base address (PSBA). This address is developed from the logical and physical PSBAR's by substituting the contents of physical PSBAR for the contents of bits 9-12 of logical PSBAR. The substitution is accomplished by shifting the data in T one position left to align logical PSBAR to a byte boundary, gating physical PSBAR, through F(0–3), into T(9–12), and then shifting T back one position to the right. The I/O unit address is gated from B(56–63) through the serial adder to S(0–7) and the channel address (B52–55), by the same route, is gated into S(20–23).

The data in S is now moved to T by gating S through the parallel adder to T. Before this is done, however, T(40–47) is gated to F to preserve bits 9–12 of the PSBA. After S has been gated to T, F is gated back to T byte 5, and the PSBA is correct again.

The microprogram now gates T to the external register, and a line called 'I/O operations' sets E(5–7) into external register positions 29–31 respectively. All data necessary for the IOCE to execute the I/O instruction is now contained in the external register.



A select, decoded from the D register, is sent to the proper IOCE, when the timing gate trigger is turned on along with STAT B. The CE must now wait for the IOCE to send back 'response' signal and a condition code. This wait period is established by loading a timeout constant in B and entering a loop which decrements B by 1 on each machine cycle. When response is received, the CE sets its condition code to the value received from the IOCE. If the timeout constant in B reaches 0 before the IOCE response is received, the CE assumes the IOCE to be inoperative and sets the condition code to 3. If a 'restart ROS timer' signal is received during the timeout, the constant is reloaded into B and the timeout is re-initiated. After the CE has set its condition code, it turns off the 'timing gate' trigger and terminates the instruction execution.

Table 3-13. Condition Code Settings, I/O Instructions

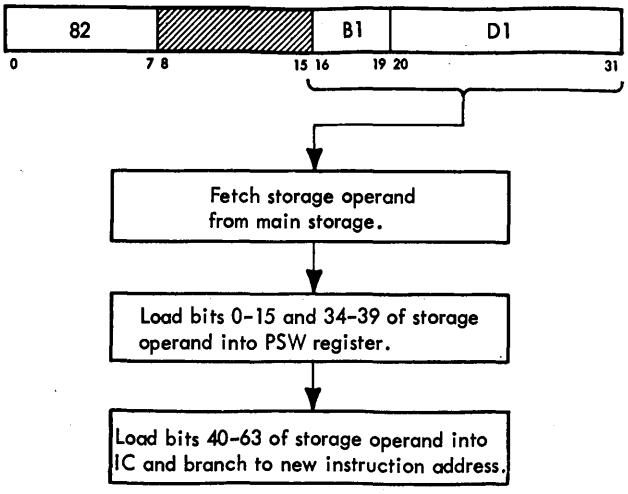
Name	Mnemonic	Op Code	Condition Code			
			Zero	One	Two	Three
Start I/O	SIO	9C	Operation initiated	CSW Stored	Channel or sub-channel busy	Not operational or invalid I/O format
Test I/O	TIO	9D	Available	CSW Stored	Channel or sub-channel busy	Not operational or invalid I/O format
Halt I/O	HIO	9E	Interrupt pending	CSW Stored	Burst operation terminated	Not operational or invalid I/O format
Test Channel	TCH	9F	Available	Interrupt pending	Channel operating in Burst Mode	Not operational or invalid I/O format
Set Program Controlled Interrupt	SPCI	9B	Not Working	CSW stored	PCI flag set	Invalid I/O address format

The nine status switching instructions can change the status of the CE's, the IOCE's, and the data in main storage:

1. The Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions control the status of the CE.
2. The Set Storage Key and Insert Storage Key instructions control the status of the data in main storage.
3. The Write Direct and Read Direct instructions control the status of the CE's and IOCE's (and also transfer data bytes).
4. The Diagnose instruction controls the status of the CE's, the SE's, and (on 9020E systems) the DE's.

LOAD PSW, LPSW (82)

- Load doubleword storage operand (designated by storage operand address) into CE, thus replacing current PSW, and branch to new instruction sequence.
- SI format:



- Use same microprogram as IPL, PSW RESTART, and interruption operations.
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.

The Load PSW, LPSW, instruction loads into the CE the doubleword from main storage designated by the storage operand address. The doubleword (a new PSW) becomes the current PSW for the next instruction. Bits 40–63 of the doubleword become the address of the next instruction. The new PSW will not allow interruptions until after the LPSW instruction is executed. When the doubleword being loaded has a 1 in position 14 or 15, the CE enters the Wait state or the Problem state, respectively; this is the only instruction available for entering these states.

Diagram 5-601, FEMDM, is a flowchart of the LPSW instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request for the storage operand has been issued per D.

Tests for a privileged operation check and a specification check are made at the beginning of the execution. The CE must be in the Supervisor state, and the address of the LPSW instruction must have three low-order 0's; otherwise, a program interruption results.

When the data (new PSW) requested during I-Fetch is available on the SDBO, it is gated to ST. The new PSW is then assembled by transferring S(0–19) and T(34–39) to the PSW register, transferring T(40–63) to the IC, and resetting the interruption request triggers. The ILC remains unchanged until an interruption occurs.

The instruction next makes a 3-cycle storage request per the IC to fill Q with the next instruction. Although the I-Fetch checking circuits have been activated previously, interruptions are inhibited by the setting of the 'I-Fetch request' trigger. The contents of T are now transferred to D so that, when the data is available, D will be able to select the correct halfword in Q to transfer to R. The IC is incremented by 8 to select the next doubleword to be fetched to refill Q. The 'PSC' and 'execute' triggers are reset. These triggers are set if the LPSW instruction is the subject instruction of an Execute instruction. Because the next instruction to be performed is determined by the new PSW being loaded, the triggers must be reset to prevent the instruction following the Execute instruction from being performed after the LPSW instruction.

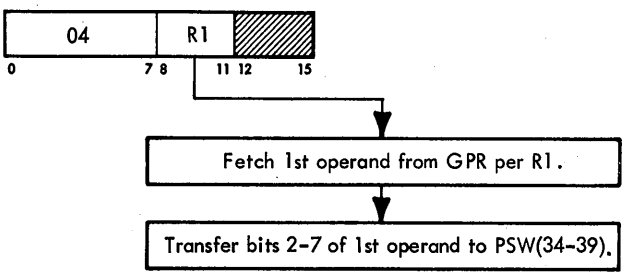
From this point on, the LPSW instruction execution is the same as for the Branch on Count instruction following a successful branch. When the requested data is available, it is gated to Q, and then transferred to R per D. A decision is now made to refill Q under control of this instruction if D(21,22) = 11. If Q does not need refilling, the instruction terminates with a branch end op. If, however, Q is refilled, a branch end op is not necessary and the instruction terminates with a normal end op.

The LPSW microprogram used by the LPSW instruction is also entered by the interruption, PSW RESTART, and IPL microprograms. The storage operand address is forced to zero by the IPL microprogram and by the depression of

the PSW RESTART pushbutton. One of the five new PSW addresses in permanent storage is generated by the interruption microprogram.

SET PROGRAM MASK, SPM (04)

- Replace CC and program mask (bits 34–39) of current PSW with bits 2–7 of 1st operand (in GPR per R1).
- RR format:

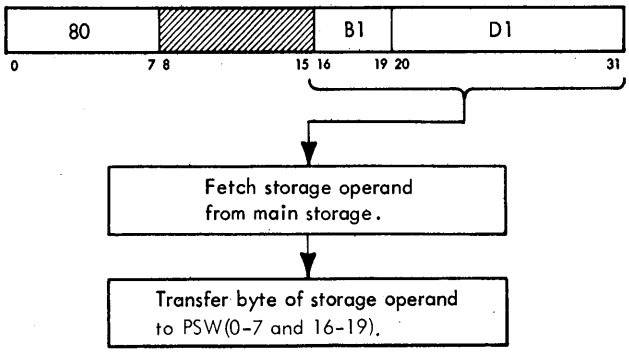


- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is not used.

The Set Program Mask, SPM, instruction replaces the CC and the program mask of the current PSW with the contents of the GPR addressed by R1 (Diagram 5-602, FEMDM). Bits 2 and 3 of the GPR become the new CC in the current PSW, and bits 4–7 become the new program mask. Bits 2–7 of the first operand may have been loaded from the PSW register by a previous Branch and Link instruction.

SET SYSTEM MASK, SSM (80)

- Replace system mask (bits 0–7 and 16–19) of current PSW with byte from location designated by storage operand address.
- SI format:

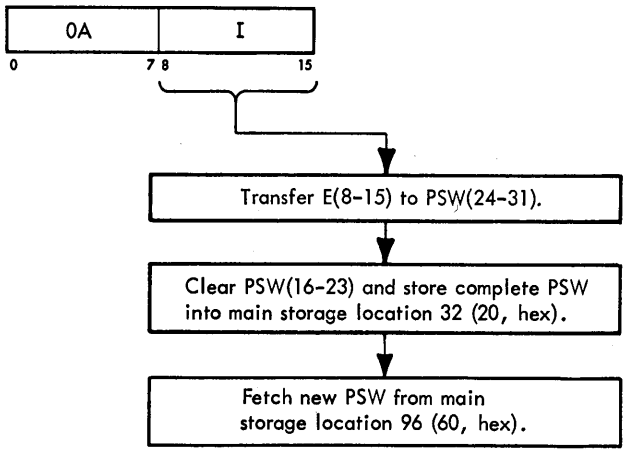


- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.

The Set System Mask, SSM, instruction replaces the system mask of the current PSW with the addressed storage operand byte (Diagram 5-603, FEMDM). The addressed storage operand byte is fetched from main storage and placed into PSW (0–7) via AB, the serial adder, and S(0–7). Bits 0–3 of the byte following the addressed byte are placed into PSW (16–19) via the same path. The ABC is set per D(21–23) to select the correct byte in AB to gate to the serial adder.

SUPERVISOR CALL, SVC (0A)

- Cause supervisor call interruption; replace old PSW (24–31) with I-field (bits 8–15) of instruction, providing interruption code.
- RR format:

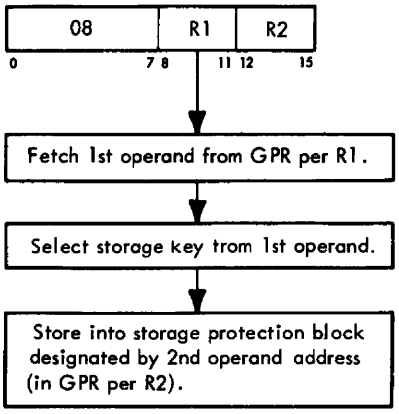


- Conditions at start of execution:
  Instruction is in E.
  E(8–15) is interruption code.

The Supervisor Call, SVC, instruction causes a supervisor call interruption at end-op (Diagram 5-604, FEMDM). The 'supervisor call' trigger is set and, if a timer exceptional condition, a machine check interruption, or a program protection interruption is not pending, the CE takes a supervisor call interruption. During that operation, bits 8–15 of the instruction, still in E(8–15), are stored as the interruption code into the supervisor call old PSW. The new PSW usually switches the CE to the Supervisor state. Refer to Section 1 of this Chapter for a discussion of the supervisor call interruption.

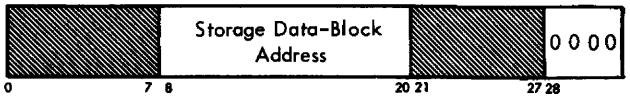
SET STORAGE KEY, SSK (08)

- Set storage key (bits 24–28 of 1st operand; in GPR per R1) for 2048-byte storage block (addressed by bits 8–20 of 2nd operand; in GPR per R2) into storage protection logic in main storage.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- Format of LS word addressed by R1:

- Format of LS word addressed by R2:



- New key is set twice because of two-way interleaving.

The Set Storage Key, SSK, instruction sets the key of the storage block addressed by the second operand according to the key in the GPR designated by R1. Bits 8–20 of the second operand address a block of 2048 storage bytes. (It is not necessary for the second operand to address the first byte in the block.) During the SSK instruction, bits 21–27 of the second operand, which address doublewords in the storage block, are ignored. Bits 28–31 of the second operand, however, must be 0's, or a program specification interruption occurs. The new storage key is obtained from bits 24–28 of the first operand; the rest of the operand is ignored.

Diagram 5-605, FEMDM, is a flowchart of the SSK instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The ABC is set to 7 (111) to allow selection of the new storage key byte from B. The STC is set to 3 (011) to select the low-order byte of the second operand in S, the byte to be tested for a specification check. D is set to the address of the storage block minus 8; this step allows construction of a loop later in the microprogram. If the system is in the Problem state, a program privileged operation interruption occurs. If the system is in the Supervisor state and the address of the storage block does not specify an even doubleword boundary [S(28–31) = 0], a program specification interruption occurs. Otherwise, the execution of the instruction continues by transferring the new key from B to F, via the serial adder.

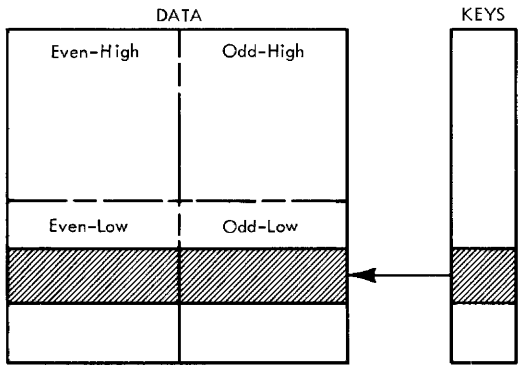
The new key is now in position to be set into the storage unit. D is incremented by 8, and a 4-cycle storage request is issued per the block address in D. The 'set key' trigger is set, causing a 'set key' signal to be sent to main storage with the 'storage select' signal, and all of the mark triggers are set. As a result, the selected storage unit recognizes its selection as being a request to change one of the keys stored in its storage protection area. The new key is gated from F to the 'key in' bus for the use of the selected storage unit.

Execution of this instruction may change the protection status of unprocessed instructions already in the CE. Therefore, the 'PSC' trigger is set to force a program store compare exceptional condition during the next I-Fetch to refill Q.

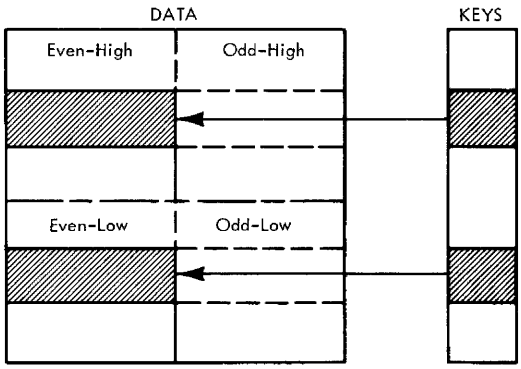
At this point, the setting of one key into main storage has taken place. When operating normally with an SE one setting of a new key for 2048 contiguous bytes of data is sufficient, as shown in Figure 3-27 (A). However, the SSK instruction in the 7201-2 always sets the key twice, first for an even doubleword address and then again for the succeeding odd doubleword address. This duplicated setting of the key is done because the CE has two-way interleaving of 'storage select' signals based on even and odd storage addresses. There is one case when contiguous even and odd storage addresses are not in the same storage-protection block in main storage:

1. When the defeat interleave maintenance aid is used, contiguous even and odd storage addresses in the 7251-9 are in two different protected blocks within the same unit, as shown in Figure 3-27 (B).

Therefore, after setting the first key, the SSK instruction again increments D by 8 and another 'set key' signal is issued to storage. For the case shown in Figure 3-27 (A), the same key location is set again; otherwise, a different key location is set.



A. SE Normal Operation.
One Key per 2048-Byte Block.

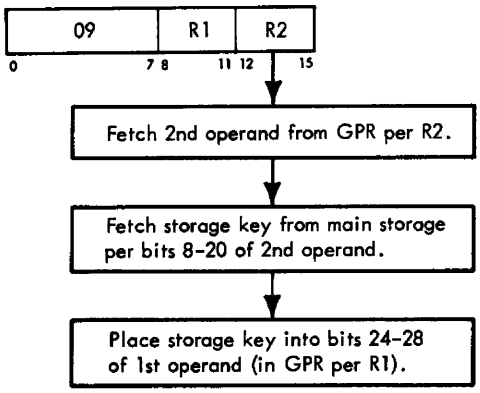


B. SE Defeat Interleave Operation.
Two Keys per 2048-Byte Block.

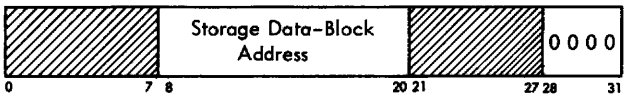
Figure 3-27. Storage Protection Key Assignments

INSERT STORAGE KEY, ISK (09)

- Insert storage protection key for 2048-byte storage block, addressed by bits 8–20 of 2nd operand (in GPR per R2), into bits 24–28 of 1st operand (in GPR per R1).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- Format of LS word addressed by R2:



- Storage key is inserted into bits 24–28 of 1st operand.
- Bits 0–23 of 1st operand remain unchanged; bits 29–31 are cleared.
- Key is fetched twice because of two-way interleaving.

The Insert Storage Key, ISK, instruction inserts the storage key addressed by the second operand into bits 24–28 of the first operand (in GPR, per R1). Bits 8–20 of the second operand address a block of 2048 bytes in main storage. Bits 0–7 and 21–27 of the second operand are ignored, whereas bits 28–31 must be 0's or a program specification interruption occurs. The five-bit storage key is set into bits 24–28 of the first operand; bits 29–31 are cleared.

Diagram 5-606, FEMDM, is a flowchart of the ISK instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The STC is set to 3 (011) in preparation of setting D to the address of the storage block minus 8, thus allowing the construction of a loop later in the microprogram. The first operand is transferred from B to T via the parallel adder, and the STC is set to 7 (111). The contents of T, with the fetched key inserted into byte 7, will later be transferred back into the GPR designated by R1. Before fetching the key, however, program tests are made. If the system is in the Problem state, a program privileged-operation interruption occurs. If the system is in the Supervisor state and the address of the storage block does not specify an even doubleword boundary [S(28–31) = 0], a program specification interruption occurs. Otherwise, the execution continues. F and the last byte of T are set to 0's. The fetching of the key can now begin.

The contents of F are logically OR'ed into the last byte of T via the serial adder, again setting 0's. D is incremented by 8 and a 3-cycle storage request is issued per the block address in D. The 'insert key' trigger is set, causing an 'insert key' signal to be sent with the 'storage select' signal. As a result, the selected storage unit recognizes its selection as being a fetch request for one of the keys stored in its storage protection area. The CE waits until a 'key advance' signal from storage is received and then gates the key from the 'key out' bus into F.

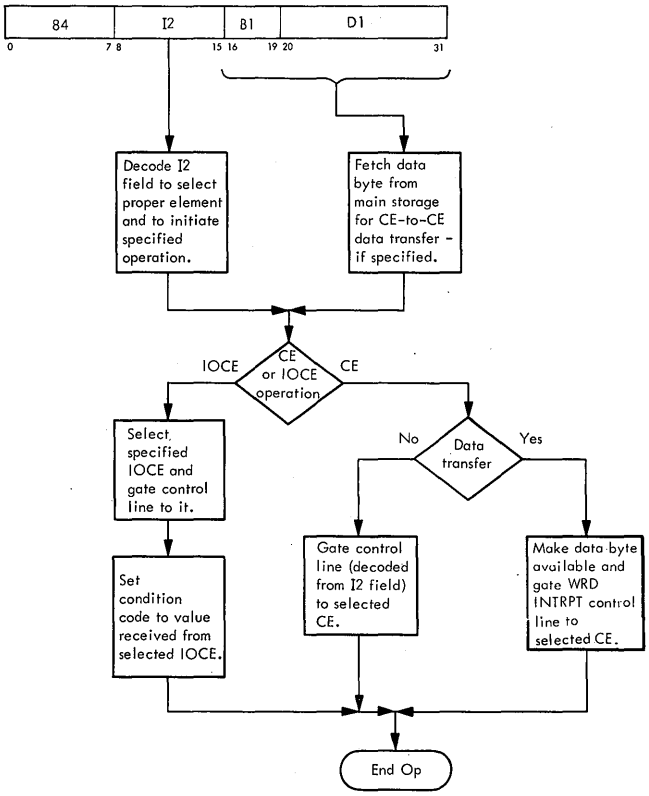
Because the key can be in two different locations, as explained for the Set Storage Key instruction, it is always

fetched twice. This scheme ensures that odd addresses can be successfully accessed. After the fetched key is logically OR'ed into T from F, the key for the next doubleword is fetched exactly as before.

When the key is again available, this time from an odd doubleword address, it is logically OR'ed with the key first fetched. The thus modified first operand is then returned to the GPR from which it came. A normal end op completes the execution of the instruction.

WRITE DIRECT, WRD (84)

- Issuing CE causes specified action in selected CE or IOCE.
- Issuing CE gates a data byte onto direct control bus or activates control lines to selected element.
- SI format:



- Condition Code:
  0 IOCE processor start or stop completed.
  1 IOCE processor start or stop completed; to/from wait state.
  2 No action taken; redundant operation.
  3 Timeout occurred; IOCE operation not completed.

- Program Interruptions:
  Privileged operation interruption if executing CE is in problem state.
  Addressing (data byte addressed is outside available storage).
  Protection (fetch protect violation).
  Specification interruption occurs if:
  1. An IOCE operation is specified and bit 15 is not 0.
  2. An IOCE operation is specified and none of, or more than one of, bits 12–14 is set to 1.
  3. A CE operation is specified and none of, or more than one of, bits 12–15 is set to 1.
  4. Bit 8 is a 1.
- For a detailed description of WRD, read the following text and refer to Diagram 5-607, FEMDM.

At the start of WRD execution, the microprogram sets ABC to the byte address contained in D(21–23). ABC is issued to select the proper byte from AB if the operation specified is a CE-to-CE data transfer. After gating the doubleword from SDBO to AB, the microprogram determines whether the operation is a CE or an IOCE operation.

In the case where an IOCE operation is decoded, the I2 field contains the control information for the WRD instruction. This field is decoded by hardware to select the designated IOCE and to cause the specified control line to be gated to the IOCE. No data can be transferred to an IOCE by WRD. One of four simplex lines (WRD INTRPT, WRD LOGOUT, EXTNL START, or EXTNL STOP) to the IOCE is activated and causes the IOCE to perform the desired action and return a condition code. The microprogram branches into a common timing routine (also used by the I/O instructions), and, as soon as a response and condition code are received from the selected IOCE, the microprogram branches to a normal end op.

Decoding bits 9, 10, and 11 of the I2 field when a CE operation is specified causes hardware to generate a select to the proper CE and to gate one of four simplex lines (WRD INTRPT, WRD LOGOUT, EXTNL START, or EXTNL STOP) to the selected CE. The microprogram in this case checks the I2 field (bits 9, 10, and 11) to see if a data byte is to be transferred. If these bits are equal to 0, the data byte is gated per ABC to the G-register. The G-register positions make up the direct control bus, and the data is active on the bus until the next WRD is issued. If I2 field bits 9, 10, and 11 are not all 0's, the operation does not involve a data transfer, and the last routine is bypassed. One of the three other simplex lines (WRD LOGOUT, EXTNL START, or EXTNL STOP) is gated to the selected CE, causing a bit to be set in that CE's external interrupt code (New PSW bits 20–31). The microprogram then branches to normal end op.

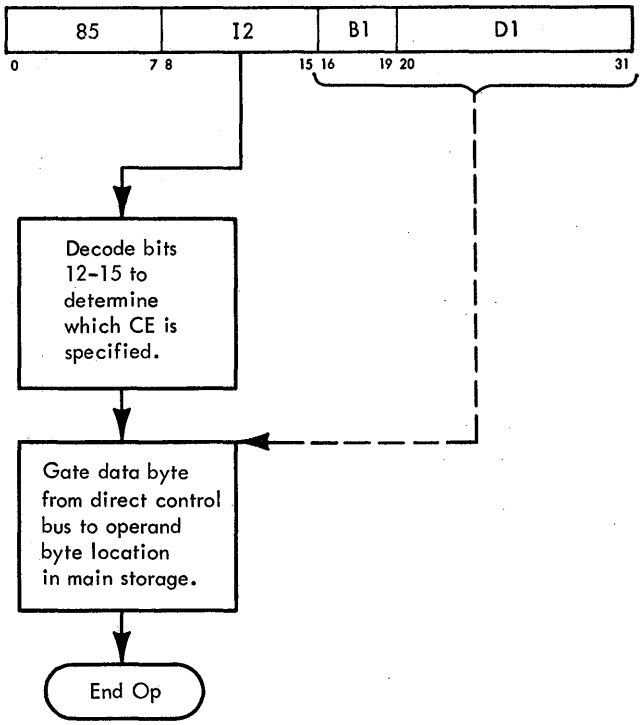
During execution of WRD, a specification error can occur if:

1. An IOCE operation is specified and more than one bit of the I2 field bits 12–14 is set to 1.
2. Bit 15 of the I2 field is a 1.
3. A CE operation is specified and more than one bit of the I2 field bits 12–14 is set to 1.
4. Bit 8 of the I2 field a 1 always causes a specification error.

Note: If an operator wishes to manually restart a CE after it has been externally stopped (Write Direct Stop), he should use the PSW RESTART pushbutton. Use of the START pushbutton may cause unpredictable results.

READ DIRECT, RDD (85)

- Gates a data byte from the direct control bus into storage at operand location.
- Causes external interrupt in issuing CE.



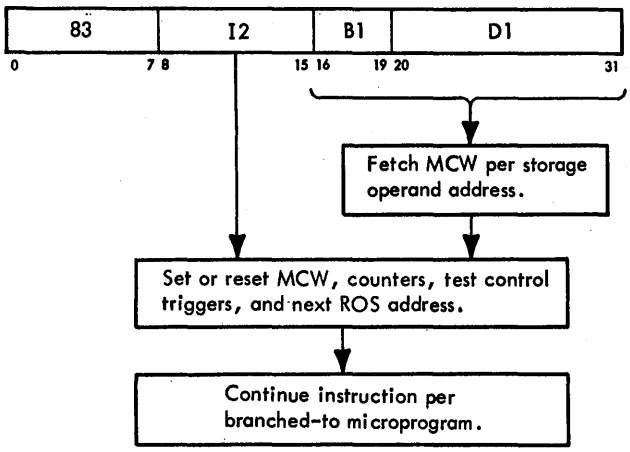
- Program Interruptions:
  Privileged operation interruption if executing CE is in problem state.
  Addressing (data byte address is outside available storage).
  Protection (store protect violation).
  Specification (more than one of bits 12–15 of the I2 field are set to 1).
- For detailed description of RDD, read the following text and refer to Diagram 5-608 FEMDM.

At the start of RDD execution, the 'hold-in' line is checked; if active, it causes the microprogram to delay for approximately 1.4 usec. Hold-in active condition indicates that data on the direct control bus could be unstable (due to another CE executing a WRD). The delay allows the CE issuing the WRD to complete its operation. If the 'hold-in' is still active after the delay, it is interpreted as an error and causes a machine check interruption.

When the 'hold-in' line is not active, the direct control bus (the read direct data byte) is gated to F. F is then gated through the serial adder into the proper ST byte per the STC. (STC was set at the start of execution to the value of D(21–23). This is the byte address (within ST) of the read direct data byte.) The mark is set per STC to store the RDD data byte in main storage. Storing is accomplished by issuing a three-cycle main storage request per D. RDD execution is then terminated by branching to a normal end op.

DIAGNOSE (83)

- Load doubleword designated by storage operand address into MCW, set or reset certain control triggers, and branch to ROS address specified by MCW.
- SI format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- Enable and disable various system maintenance aids.

The Diagnose instruction has two purposes: it is available to the diagnostic programmer as a maintenance aid and is available to the system programmer for 9020 mode operations. In both applications, the beginning of its execution is basically the same. The immediate operand is used to set or reset control triggers. The storage operand, a doubleword termed the maintenance control word (MCW), is used to set the MCW register, the counters, and the address of the next ROS word. The remainder of the

execution is determined (1) by the ROS microprogram branched to and (2) by the control triggers.

Diagram 5-609, FEMDM, is a flowchart of the Diagnose instruction. At the start of execution, the first 16 bits of the instruction are in E, the address of the storage operand is in D, and a storage request for the storage operand has been issued per D. If the address of the storage operand does not specify a doubleword boundary, a program specification interruption occurs. At this point, the address of the storage operand is no longer needed and is incremented by 8.

If the CE is in state 0, it is placed into the Scan mode, and the MCW is gated from the SDBO(0–31) to T (and also SDBO 0–63 to AB). The control triggers are not set or reset: T(32–39, 52) and B(32–51) are transferred to the MCW register, T(53–57) to the address sequencer, T(58–61) to the FLT counter, T(62, 63) to the FLT clock, E(8, 9) to the two interleave control triggers, and E(10) to the 'diagnose FLT' trigger. The 'scan counter control' and 'diagnose' triggers are set. They remain set depending on the application. If the CE is in state 3, 2, or 1, it is placed into Scan mode, and the ROS branch address (bits 8–15) of the MCW is checked for FD (hex). If any other value is found, Scan mode is reset and a specification error is set, forcing a program interruption. If the address bits 8–15 are found to contain FD (hex), the I2 field and all MCW bits are degated except bits 32–35 and 50, and address bits 8–19. The CE is now taken out of Scan mode, and a ROS branch is taken.

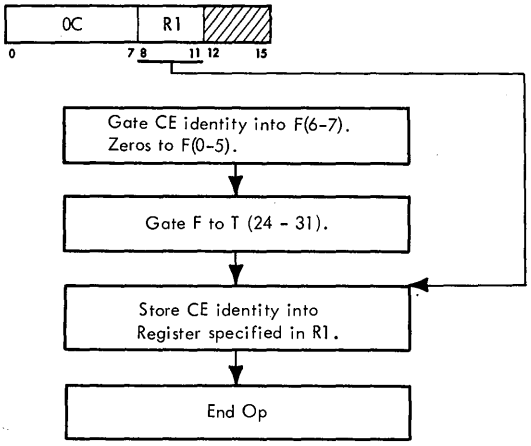
The ROS branch address is transferred from T(40–51) to ROSAR(0–11). If the instruction is being used to reset a previously set trigger, the address of a normal end op, 010 (hex), is usually specified, and the execution is completed. However, any ROS address may be specified, depending on the application.

SECTION 9. MULTIPLE COMPUTING ELEMENT INSTRUCTIONS

This section discusses the ten instructions that make up the multiple computing element instruction set. The need for this instruction set develops when multiple computing elements must operate simultaneously, without conflict, in a multiple element shared storage environment such as the 9020D/E system. These instructions provide for system configuration, storage assignment, and preferential storage area assignment in the 9020D/E system.

LOAD IDENTITY, LI (0C)

- Loads the identity of the CE executing the instruction into bits 28–31 of the GPR specified by the R1 field.
- RR Format:



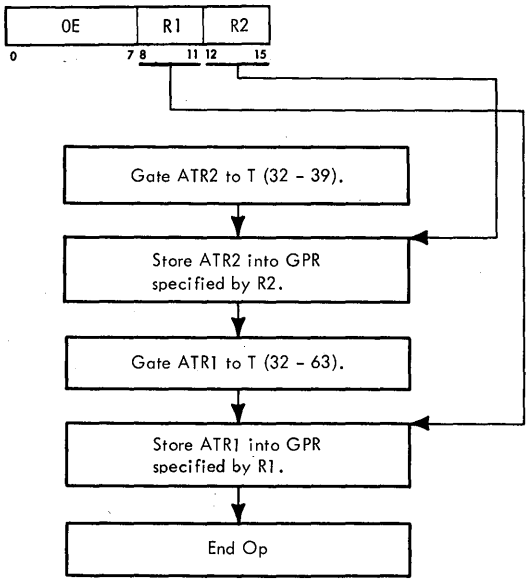
- Enables programmer to identify the CE executing the program.
- Condition code: unchanged
- Program interruptions: none
- Refer to Diagram 5-801, FEMDM.

A two-bit fixed point integer code is assigned to each CE. The CE with the highest priority for accessing storage (CE1) is assigned identity 0; that with the lowest priority (CE4) is assigned identity 3.

The assignment is made by wiring a jumper card in the CE. In the execution of the LI instruction, the output of this card is gated into F(4–7). F is then gated through the parallel adder into T(60–63), and from T into the GPR specified by the R1 field.

INSERT ATR, IATR (0E)

- Places the contents of ATR1 and ATR2 into two GPRs specified by the R1 and R2 fields.
- RR Format:



- Condition code: unchanged
- Program interruptions: none
- Refer to Diagram 5-802, FEMDM.

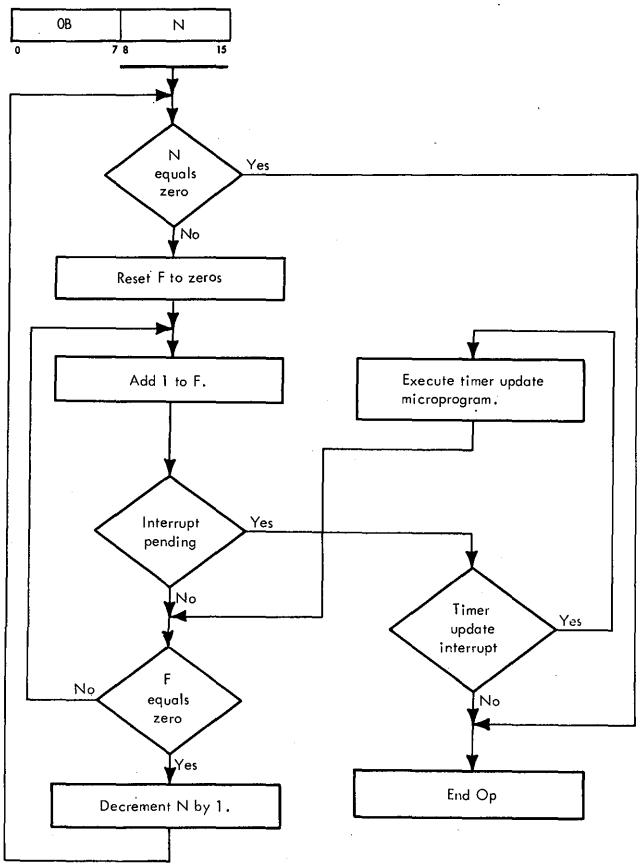
ATR2 is gated via the 9020 out bus and the LS output bus into S(0–7). At the same time, 0's are set into S(8–31) since there is no information on the LS output bus for these positions.

S is then moved to T via the parallel adder, and the T contents are written into bits 0–7 of the GPR specified by the R2 field.

Next, ATR1 is gated directly to T(32–63) via the 9020 out bus and the LS output bus. This information is written into bits 0–31 of the GPR specified by the R1 field, and the instruction execution is terminated.

DELAY, DLY (0B)

- Provides a variable delay dependent on value (N) specified by bits 8–15 of the instruction.
- RR Format:



- Terminated by the count (N) reaching zero or by any interrupt except a timer update.
- Condition code: unchanged
- Program interruptions: none
- Refer to Diagram 5-803, FEMDM.

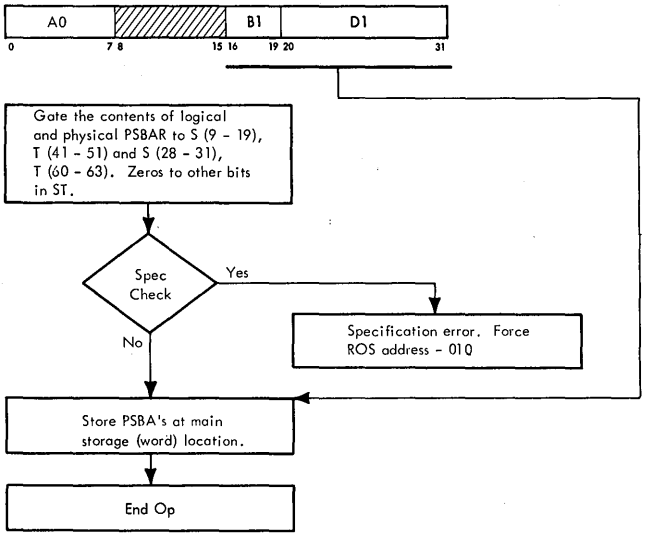
The F register is used as a counter during the execution of the instruction. F is incremented by 1 every microsecond until its count equals 256. If E(8–15) is not 0 when the count reaches 256, 1 is subtracted from the value N, in E(8–15). If this subtraction does not reduce N to 0, F is reset, and the operation is repeated. As soon as N equals 0 the instruction operation is terminated.

A check for pending interrupts is made after every addition to F. All interrupts except a timer update cause the termination of the instruction at that time.

Timer update interruptions are executed, and control is then returned to the Delay instruction. Thus, requests for service by the interval timer may affect the actual delay obtained.

STORE PREFERENTIAL-STORAGE BASE ADDRESS REGISTER, SPSB (A0)

- Store contents of Logical PSBAR in bit positions 9–19 at the word location in main storage (specified by B1 + D1).
- Store contents of Physical PSBAR in bit positions 28–31 at the word location in main storage (specified by B1 + D1).
- Bit positions 0–8 and 20–27 at the word location in main storage (specified by B1 + D1) are set to 0's.
- SI Format:



- Program interruptions:
  1. Specification interruption (occurs if operand not specifying a word boundary).
  2. Priviledged operation interruption.
- Condition code: unchanged
- Refer to Diagram 5-804 FEMDM.

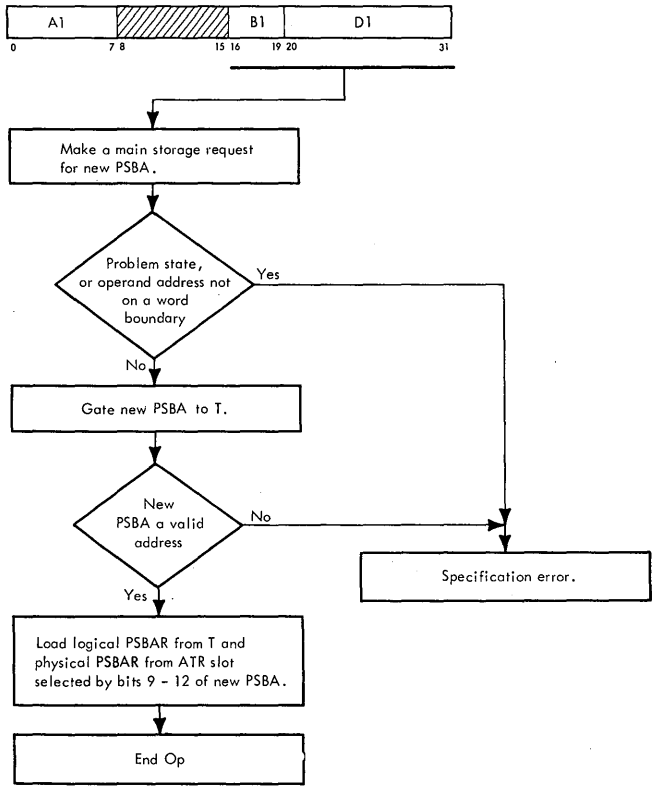
The contents of Logical and Physical PSBAR are stored, in bit positions 9–19 and 28–31, respectively, at the operand location in main storage. The unused bit positions at this location (0–8 and 20–27) will contain 0's. A specification exception occurs if the operand location is not on a word boundary.

At the beginning of execution, Logical and Physical PSBAR are gated, via the 9020 out bus and the LS output bus, to S(9–19 and 28–31) and T(41–51 and 60–63). Since nothing is gated to the output bus bits 0–8 or 20–27 these positions in ST become 0's.

The correct marks are set per D (bit 29), and the word is stored if no errors were detected. The instruction execution is terminated after an address store compare test is made.

LOAD PREFERENTIAL-STORAGE BASE ADDRESS, LPSB (A1)

- Loads the logical preferential storage base address register (PSBAR) from a storage location pointed to by the operand.
- Loads the physical preferential storage base address register (PSBAR) from the ATR slot selected by bits 9–12 of the newly fetched Logical PSBA.
- SI Format:



- Program interruptions:
  1. Specification Interruption occurs if:
     a. Main storage address is not specified on a word boundary.
     b. New PSBA designates a location that is not in a properly configured SE.
  2. Privileged operation interruption occurs if executing CE in problem state.
- Condition code: unchanged
- Refer to Diagram 5-805, FEMDM.

At the start of execution, the microprogram checks for errors that can cause the instruction to be terminated. A specification error is set if the main storage address is not specified on a word boundary. A privileged operation error occurs if the executing CE is in the problem state.

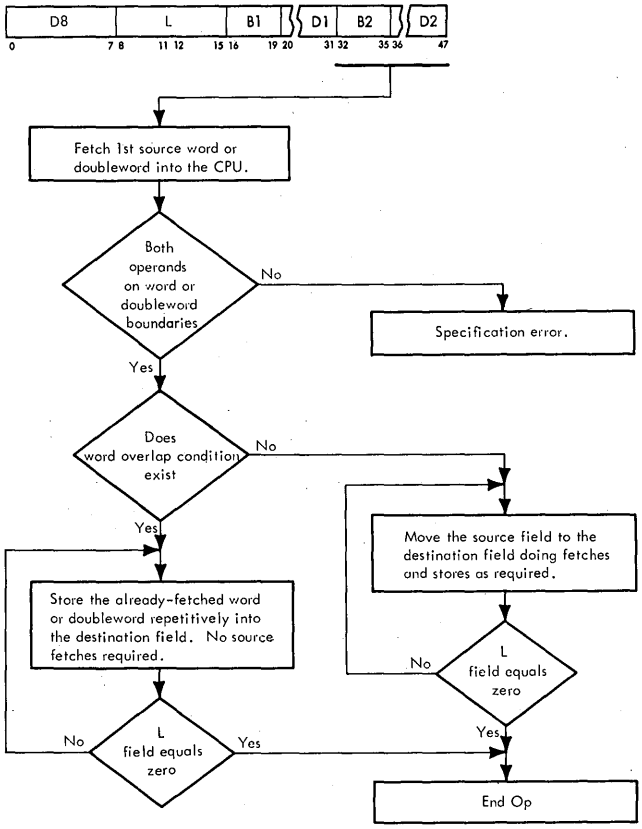
The new preferential storage base address (PSBA) is fetched into S(8–19) by making a storage request per D. The possible problem created by extraneous bits in the unused 4 bits of T(48–55) is overcome by setting T(52–55) to 0's. T(48–51) are gated through the serial adder along with 0's for the remaining bits, parity is corrected, and the result is set back into T(48–55).

The new PSBA must be a valid address. This is checked by gating T(40–63) to D and making a main storage request per D. If this request causes an invalid address, a specification error is set. If there is no invalid address indication, the Logical PSBAR is loaded from T(41–51). Logical PSBAR (9–12) are used to select an ATR slot. The contents of this slot are set into the Physical PSBAR.

MOVE WORD, MVW (D8)

- Moves up to 256 words from one storage location (B2 + D2) to another storage location (B1 + D1).
- Number of words to be moved indicated in the L field.
- Program interruptions
  1. Protection (store or fetch protect violation).
  2. Addressing (data or part of data is outside of available storage).
  3. Specification (either operand specifies other than a word or doubleword boundary).
- Condition code: unchanged
- Refer to Diagram 5-806 FEMDM.

- SS Format:



This instruction moves up to 256 words from one storage location (the source field-B2 + D2) to another storage location (the destination field-B1 + D1). Movement through the fields is from left to right, and the number of words moved is one greater than the value specified in the L field.

Fetches and stores from SEs and DEs are always on doubleword boundaries. To align the data when both fields are not on doubleword boundaries, the instruction must manipulate the source data so that it is stored correctly in the destination field.

There are four possible combinations of the source and destination address alignment (each case requires a separate move word routine): Case A: Source field on a doubleword boundary and destination field on a doubleword boundary. Case B: Source field on a word boundary and destination field on a word boundary. Case C: Source field on a doubleword boundary and the destination field on a word boundary. Case D: Source field on a word boundary and destination field on a doubleword boundary.

If a word overlap condition exists, the routines are modified to save execution time. The word overlap condition occurs when the starting address of the destination field is one word or one doubleword higher than the starting address of the source field. This condition propagates the first source word (or doubleword) through storage for the number of words specified in the length field. Once the word or doubleword of source data has been fetched by the CE, further fetches are not necessary, and repeated stores can be made per the destination address. If a word overlap condition is detected, STAT A is set for subsequent modifications to the move-word routines.

The first doubleword of source data is fetched into S and T at the beginning of execution. Then, the AB and ST counters are used to check that both operands are on word or doubleword boundaries. The check is made by gating the three low-order bits of the source address into ABC and the three low-order bits of the destination address into STC. If the operand address is on a doubleword boundary, the three low-order bits of the address will equal 0; if on a word boundary, these address bits will be equal to 4. The values in ABC and STC are decremented by 1 to obtain values of: 7 (if doubleword boundary), or 3 (if word boundary). This subtraction is done to utilize the existing micro-orders which branch when the value of ABC/STC is equal to 3 or 7 (i.e., a direct branch on ABC/STC equal to 4 or 0 is not possible). A specification error is set if the counters do not equal 3 or 7 after subtraction. In addition to the checks for a specification error, the AB and ST counters determine which of the four move word routines is required.

Case A: Source and Destination on Doubleword Boundary

The first source operand was fetched into S and T at the beginning of execution. If the word count is 0, only the first word of the source doubleword is stored, and the instruction execution is terminated. However, if the word count is not 0 the previously fetched source doubleword is stored in the destination field, and, if word overlap is present (STAT A set), the high-speed move routine is entered. This routine makes no source fetches. It simply stores the doubleword in S and T per the destination address (in D), updates the destination address, and decrements the word count. This sequence is repeated, and, when the count equals 0, the specified number of words have been moved and the instruction execution is terminated.

Case B: Source and Destination on Word Boundary

The first word of the source field was previously fetched, and is in T at the beginning of this routine. This word is stored at the starting address of the destination field. If there is word overlap (STAT A set), the next doubleword of source data is fetched (per IC), and the high-speed move routine is entered. As in case A, the source data in S and T is stored per the destination address, the destination address

is updated, and the word count is decremented. When the word count reaches 0 the instruction is terminated.

If word overlap does not exist, new source data must be fetched after each store into the destination field. The source field is moved to the destination field by repetitive fetches and stores, during which both source and destination field addresses are updated, and the word count is decremented. The instruction execution is terminated when the word count reaches 0.

Case C: Source on Doubleword Boundary, Destination on Word Boundary

If word overlap exists, T is stored in the destination field, and the destination field address is updated. S and T are then stored repetitively per the destination address. When the word count, decremented 2 for each store cycle, reaches 0, the instruction execution is terminated.

In the case of no word overlap, the first source word (in S) is to be stored on a word boundary, and must be moved to T. Thus, the second source word (in T) must be moved to the LSWR so that S can be gated to T and stored (on a word boundary) at the destination field starting address. The next source doubleword (source words 3 and 4) is then fetched into S and T. Source words 2, 3, and 4 are now in the CE (2 in LSWR, 3 in S, and 4 in T). Words 2 and 3 must be in S and T, respectively, before they can be stored in the destination field. This rearrangement is accomplished by gating S (word 3) to PAL and holding it there for an extra cycle. The LSWR (word 2) is gated to S, and T (word 4) is gated to the LSWR. PAL (word 3) is gated to T, and now source words 2 and 3 are in the correct position to be stored into the destination field. Word 4 is saved in the LSWR until the next two source words are fetched into the CE. Then, the same rearrangement takes place, and the next two sequential words are stored in the destination field. This sequence continues, and the word count is decremented accordingly. When the word count equals 0 the instruction execution is terminated.

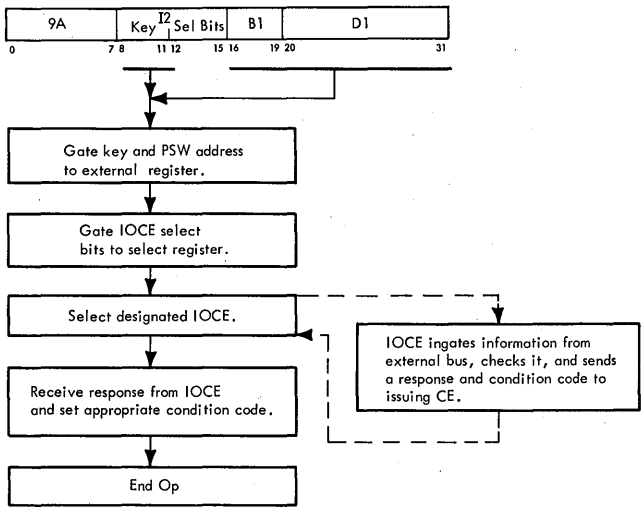
Case D: Source on Word Boundary, Destination on Doubleword Boundary

Source word one is in T (and in LSWR) at the beginning of this routine. If word overlap exists, the LSWR is gated to S, and the high-speed move routine is entered. No source fetches are required, and source word one is propagated through the destination field by storing doublewords. The word count decrementing to 0 terminates the instruction execution.

Where word overlap does not exist, this routine is very similar to the one for Case C. Source word 1 is in the LSWR, and, after words 2 and 3 have been fetched, the same rearranging must be done. Source words are fetched, rearranged, and stored in the destination field. The word count is decremented until the specified number of words have been moved.

START I/O PROCESSOR, SIOP (9A)

- Transfers storage key and PSW address to the selected IOCE.
- SI Format:



- Program interruptions:
  1. Privileged operation interruption, if executing CE is in problem state.
  2. Specification interruption if:
     a. E register bit 15 is a 1.
     b. More than one bit is on in E (12–14).
     c. A key of F (hex) is specified.
     d. PSW address is not on a doubleword boundary.
- CC Setting:
  PSW has been loaded successfully, and the IOCE-processor is proceeding with its execution: CC = 0
  PSW is invalid: CC = 1 (CC = 2 is not used)
  Selected IOCE not operational (no response received.): CC = 3
- Refer to Diagram 5-807 FEMDM.

The SIOP sends (to the designated IOCE) the address of a PSW and the storage protect key for that location. The IOCE responds to the CE and sends a condition code after checking the key and address for validity. The IOCE then performs a load PSW and begins the designated operation.

The I2 field in the instruction format is divided into two parts: the key (bits 8–11), and the IOCE select mask (bits 12–15). The MS address, specified by the B1 and D1 fields, is the PSW address, which is sent to the IOCE along with the key.

The PSW address is calculated and placed in D during I-fetch. This address is now gated through the parallel adder into T(36–59). The key is gated from E(8–11) into B(60–63). The key is then aligned in front of the PSW address in T by cross-gating B(60–63) and T(36–39) through the serial adder. The key is now in T(32–35), adjacent to the PSW address in bits 36–59. This information is gated from T to the external register, and from there it goes onto the external bus; then the specified IOCE is selected.

The IOCE select mask is transferred to the select register by gating E(12–15) through the parallel adder into T(60–63) and, then, gating T to the select register.

The select is sent to the proper IOCE by turning on STAT B and the timing gate trigger. The CE must now wait for the IOCE to send back 'response' signal and a condition code. This wait period is established by loading a timeout constant in B and entering a loop routine which decrements B by 1 on each machine cycle. When response is received, the CE sets its condition code to the value received from the IOCE. If the timeout constant in B reaches 0 before IOCE response is received, the CE assumes the IOCE to be inoperative and sets the condition code to 3. After the CE has set its condition code, it turns off the 'timing gate' trigger and terminates the instruction operation.

SET ADDRESS TRANSLATOR, SATR (0D)

- Discussion of SATR instruction is presented in three parts:
  1. Introduction to address translation.
  2. Instruction execution in the issuing CE.
  3. Action initiated by SATR select in a receiving CE.

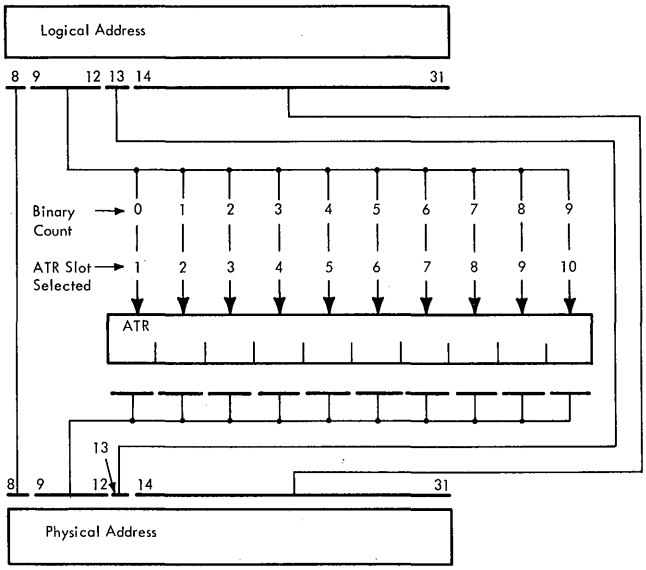
Introduction to Address Translation

- Allows any SE in the system to be assigned to any 512,000 block of the available storage addresses.
- Replaces bits 9–12 of the programmer's logical address with the four bits of an ATR slot to obtain the physical address.
- Address translation is active for all addressing in the 9020 system.

Address translation is defined as the conversion of a programmer's logical address into the machine's physical address. Using the Set Configuration (SCON) and the Set Address Translator (SATR) instructions, the 9020 system can alter its configuration to replace failing elements or to respond to additional system loads.

The SCON instruction defines the system and sets up the communication paths by setting the configuration register in each element. (For additional details, refer to the discussion of SCON.) The SATR instruction allows the programmer to arrange the storage elements within the defined system in any physical order that he chooses. For example, a system having SE 2 and SE 5 configured has a total address range of 1,024,000 bytes. Addresses 0 to 511,999 can be assigned to SE 2 (by loading ATR slot 1 with a 2), and addresses 512,000 to 1,023,999 can be assigned to SE 5 (by loading a 5 in ATR slot 2). The reverse can also be done by assigning addresses 0 to 511,999 to SE 5 (loading 5 into ATR slot 1) and 512,000 to 1,023,999 to SE 2 (loading 2 into ATR slot 2).

Bits 9–12 of the address specify, in 0.5 M-byte increments, a logical address range from 0 to 8 M-bytes. Without address translation, the logical and physical addresses would be the same, and address bits 9–12 would be used to select the 0.5 M-byte storage elements. Address translation in the 9020 system uses logical address bits 9–12 to select a slot of the address translation register (ATR). This slot (loaded by the SATR instruction) contains the physical storage element identifier for the storage frame addressed by the logical address. Logical address bit 8, the four bits of the ATR slot, and logical address bits 13–31 are combined to form the machine's physical address.



The purpose of the SATR instruction is to reassign storage frames by loading the ATRs of the designated CEs (and IOCEs) with new physical addresses. The CE executing the SATR instruction places the contents of two GPRs

(specified by the R1 and R2 fields) onto the external bus and issues selects to the designated CEs and IOCEs, causing them to gate the bus into their address translation registers. The two GPRs are loaded by the programmer with the new ATR information. The first GPR (specified by R1) holds ATR slots 1–8. The second GPR (specified by R2) holds slots ATR 9 and 10 and the selection mask (24–31); bits 16–23 of this GPR are reserved for the response byte. At the completion of the SATR instruction, the response byte is set to indicate those elements that failed to respond.

Instruction Execution in the Issuing CE

- Loads ATR of each CE and IOCE specified by the selection mask. ATR slots 1–8 are loaded from GPR specified by R1. ATR slots 9 and 10 are loaded from bits 0–7 of GPR specified by R2. Selection mask is in bits 24–31 of GPR specified by R2. (9020D and 9020E systems have different ATR slot assignment format.)

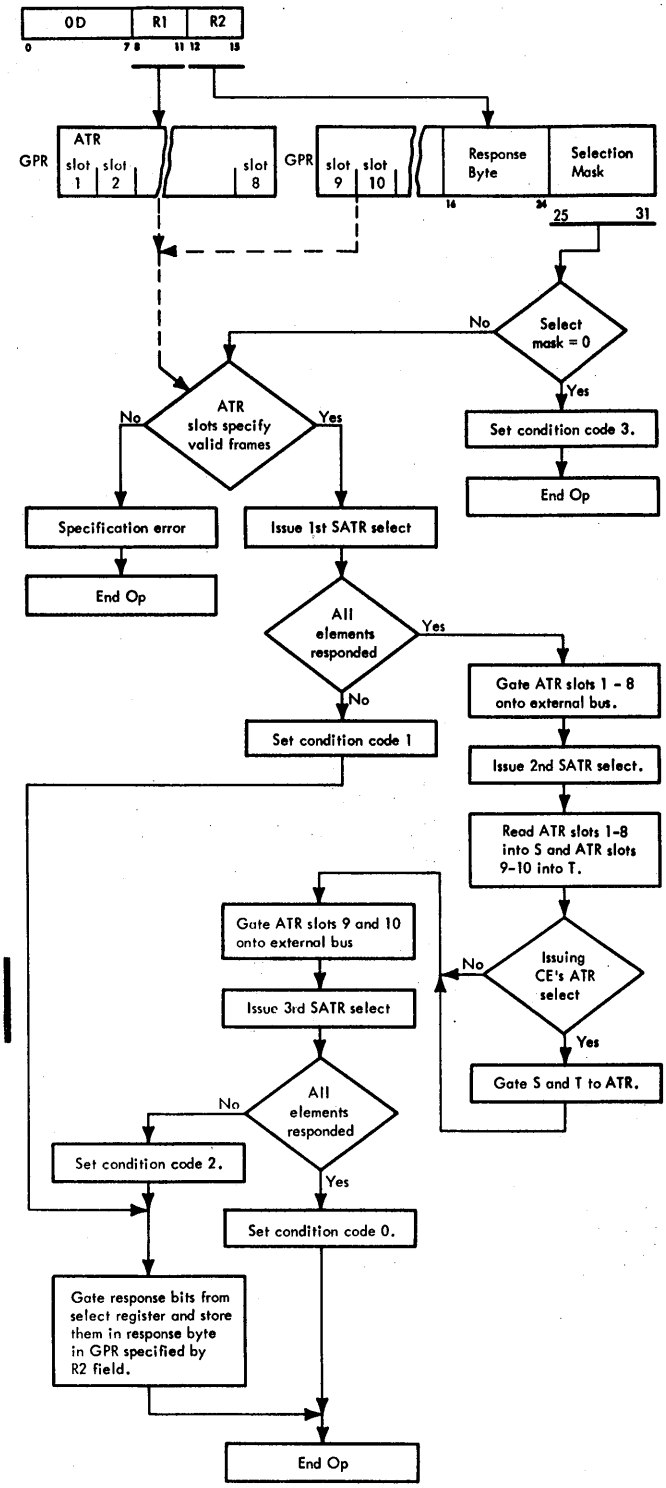
- During execution, the 'SATR select' signal to the receiving elements is activated at three different times:
  1. 1st 'SATR select' initiates responses from selected CEs/IOCEs.
  2. 2nd 'SATR select' ingates ATR slots 1–8 from the external bus into the ATRs of selected CEs/IOCEs.
  3. 3rd 'SATR select' ingates ATR slots 9 and 10 from the external bus into the ATRs of selected CEs/IOCEs.

- RR Format: (See Adjacent Column)

- Program interruptions: A specification interruption occurs if:
  1. The CE executing the instruction does not have its own SCON bit set in its CCR.
  2. The CE executing the instruction is not in state 0 or 3.
  3. The ATR assignment mask specifies an invalid frame (one that is not available to the particular installation).
  4. On a 9020E, the ATR assignment mask assigns an SE to a position reserved for a DE, or a DE to any position other than 6, 7, 8, 9, 10.

  A privileged operation interruption occurs if the CE executing the instruction is in problem state.

- Condition code:
  All selected elements have accepted new ATR information: CC = 0.
  One or more selected elements failed to respond to the 1st SATR select. (This occurs if the issuing CE's SCON bit was not on in the selected element's CCR.): CC = 1.



One or more selected elements failed to respond to the 3rd SATR select. (This occurs if the selected elements detect bad parity in the new ATR information.): CC = 2.
Selection mask in the GPR specified by R2 is equal to 0: CC = 3.

- Refer to Diagram 5-808 FEMDM.

At the start of execution, the selection mask is checked for all-0's. If this condition exists, no elements have been selected; the condition code is set to 3, and the instruction is terminated.

The instruction then proceeds to check the ATR slot assignment for incorrect specification. If an ATR slot is specified incorrectly, a specification error is detected, and the instruction is terminated.

In the 9020D system, all ATR slots are assigned to SEs. For this system, the ATR slot is invalid if:

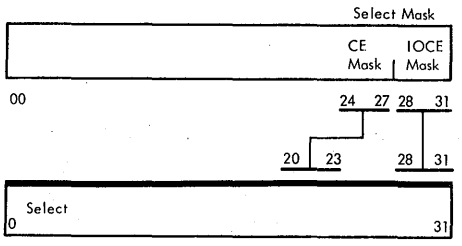
1. A frame is decoded which is not in the system.
2. Frame identifiers B through F (hex) are specified. (These identifiers are illegal because there can be a maximum of 10 frames (A-hex) in the system.)

The 9020E system ATR slots 1–5 must specify SEs, and ATR slots 6–10 can only specify DEs. For this system, the ATR slot is invalid if:

1. A frame is decoded which is not in the system.
2. A DE is specified in ATR slots 1–5.
3. An SE is specified in ATR slots 6–10.
4. Frame identifiers B–F (hex) are specified.

The microprogram, first, checks the validity of ATR slots 1–5 and, then, 6–10. The ATR slots are selectively transferred from AB to F for validity checking. Note that each time a pair of ATR slots is gated from AB (per ABC) to the serial adder bus, STC is used to select the correct ATR slot within the pair. This slot is gated by micro-orders to F(0–3). STAT C is set to indicate the validity of the ATR mask. If STAT C remains set after all the ATR slots have been checked for validity, the ATR mask is valid, and execution continues. If STAT C is off, a specification error is set, and the instruction is terminated.

After checking the validity of the ATR slots, the instruction transfers the selection mask from B to the select register. Note that the select register bit position assignments for CEs and IOCEs are different from the original SATR select mask bit positions in the GPR. Therefore, the select mask (in B) must be split and gated to the proper bit positions in the select register.



On its way to the select register, the select mask, in the proper select register format, is set into S and saved for later use.

After the select register is loaded, the first SATR select is activated by turning on the 'timing gate' trigger and ANDing it with: (1) STAT A and (2) the select register bit for each element. This select is held active for approximately 25.6 usec by a delay loop, which uses the F register as a counter. The 25.6-usec delay allows any operations in progress to complete.

After 25.6 usec, the 'timing gate' trigger is turned off to allow the response lines from the selected elements to reset the corresponding select register bits. A 2-cycle delay allows responses from the elements to reset the select register bits. The select register is then gated (via T) to the parallel adder for an all-0's check. The condition when all units respond and are waiting to ingate the new ATR information from the external bus is indicated by the PAL = 0.

The select mask (in S) is gated back into the select register, and ATR slots 1–8 (in A) are placed in the external register. The 'timing gate' trigger is turned back on, to issue the second SATR select. Upon receipt of the second SATR select, the selected element ingates information from the external bus into its ATR. A delay loop using the AB counter holds the second SATR select up for approximately 4.4 usec. No response is expected from the elements.

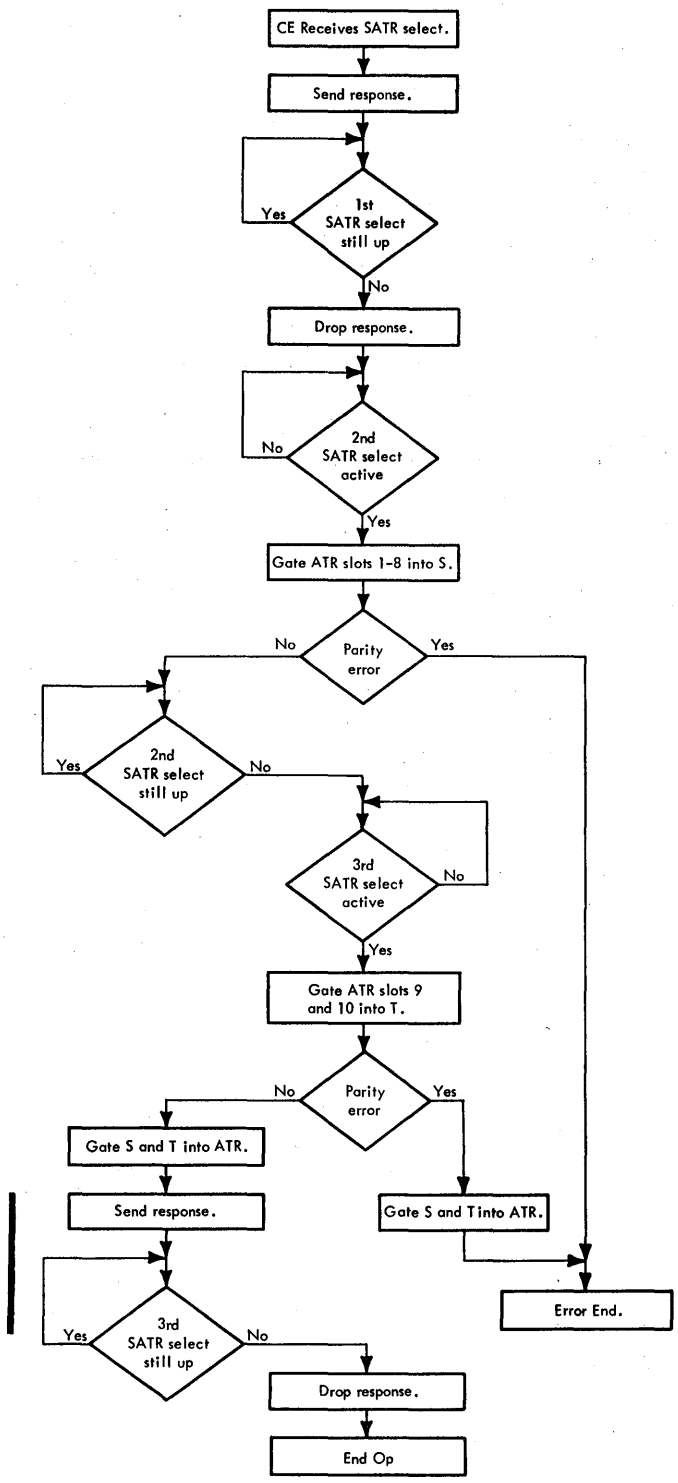
If the CE executing the SATR instruction has to set its own ATR, the ATR is set after the drop of the second SATR select. This is accomplished by gating slots 1–8 to S and slots 9 and 10 to T(32–39). ST is then gated to the ATR.

The third SATR select signal is activated by turning on the 'timing gate' trigger after slots 9 and 10 (in T) have been placed in the external register. A delay loop, using A as a counter, holds this select up for approximately 9.6 usec. A 2-cycle delay allows responses from the elements to reset the select register bits. The select register is then gated to the parallel adder (via T) for an all-0 check. The 'timing gate' trigger is turned off, and the instruction execution is terminated if PAL is 0. The micro-program sets the condition code to 0 to indicate that all elements responded and the ATRs were set with no errors.

The response mask is saved and stored in the response byte location of the GPR specified by the R2 field. The CE and IOCE bits, located in different bytes in the select register, are gated to T via the 9020 out bus and the LS out bus. The CE and IOCE response bits are combined and written into the GPR specified by the R2 field: select bits 20–23 to GPR bits 16–19, and select bits 29–31 to GPR bits 21–23.

Action Initiated by SATR Select in a Receiving CE

- A CE receiving a SATR select signal from another CE executes a microprogram to ingate the new ATR information. The receiving CE must keep itself synchronized with the issuing CE. This is done by monitoring the rise and fall of the SATR select line sent by the issuing CE.



- Refer to Diagram 5-809, FEMDM.

When the CE receives SATR select, the 'time clock step' trigger is turned on to force an interrupt to the operation in progress. Also turned on are an input latch for the issuing CE (CE 1, 2, 3, or 4 'input' latch) and the 'external SATR select' latch. The SATR response line, for the first select received, is sent back to the issuing CE by turning on the 'interrupt gate' trigger. The receiving CE must wait now for the first select to drop. For this purpose, a timeout loop using the F register as a counter is set up. The drop of SATR select ends the timeout, and the microprogram turns off the 'interrupt gate' trigger. A new timeout loop is started to wait for the second SATR select to become active.

With the arrival of the second SATR select, the receiving CE gates the external bus (ATR slots 1–8) into S via the 9020 and local store out buses. This information is parity-checked while it is on local store out bus. An error at this time forces the receiving CE into the machine check microprogram. Therefore, a response to the third SATR select cannot be sent. This informs the issuing CE of the parity error.

Another timeout loop is set up to wait for the second SATR select to drop. As soon as select drops, another timeout loop is set up since all that must be accomplished is to wait for the third SATR select to be issued.

When the third SATR select is received, the external bus (ATR slots 9 and 10) is gated to T, again via the 9020 and local store out buses. Data is checked on the local store out bus. The new ATR information must be set into the receiving CE's address translation register. This is accomplished by gating S(00–31) to ATR (00–31) and T (32–39) to ATR (32–39). The receiving CE has now set its own ATR. To inform the issuing CE of this, if there are no parity errors, the receiving CE raises SATR response (by turning on the 'interrupt gate' trigger).

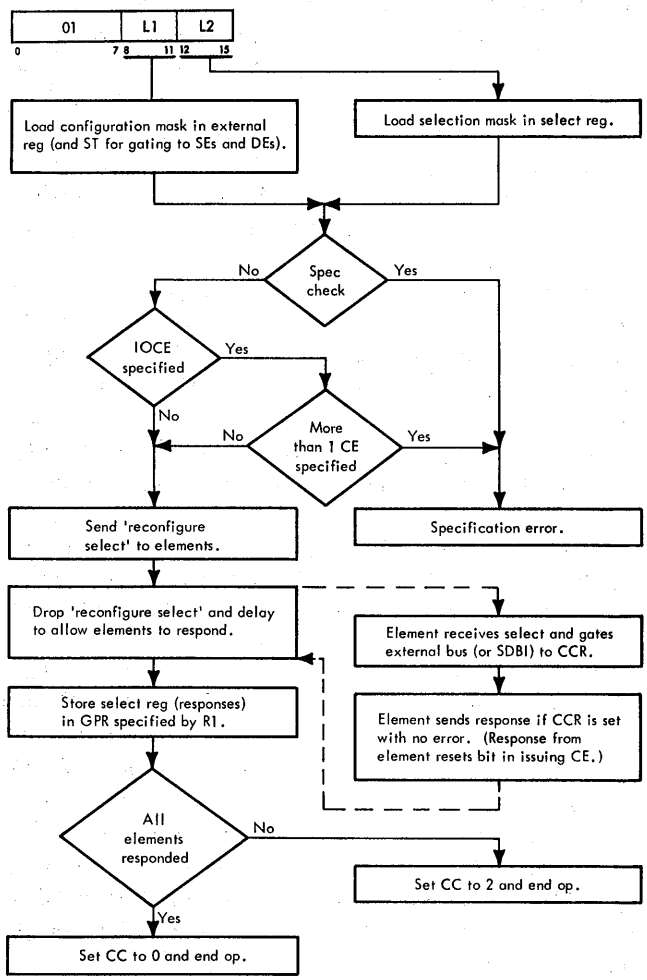
If there is a parity error on transfer of slots 9 and 10 ATR information to the T register, 'response' is not issued, but the 9 and 10 portion of the CE's new ATR contents will be destroyed during error logout. However, the error ATR word will be available in the T register logout word. Only the original ATR information as it existed prior to the SATR instruction is lost. With no error, a new timeout loop is entered to wait for the drop of the third SATR select. As soon as select drops, the 'interrupt gate' trigger is turned off.

At this point, the microprogram enters previously used timeout loops. The microprogram will now branch on the timeout condition (SAL = 2F hex) because the SATR select line from the issuing CE will neither rise nor fall. When timeout occurs, the CE resets the 'external SATR select' latch, the proper 'CE input' latch, and the 'time clock step' trigger, and goes to end op.

SET CONFIGURATION, SCON (01)

- Allows a CE to select one or more elements and to place configuration mask bits into their CCRs.

- RR Format:



- Select mask specifies which elements are to be configured.

- Configuration mask specifies to the selected elements:
  1. The state they are to assume.
  2. The CE(s) from which they can accept future SCON instructions.
  3. The elements with which they can communicate.

- Configuration mask is sent to SEs and DEs, via the SDBI, and to all other elements, via the external bus.

- A privileged operation interruption occurs if the CE attempts to execute the SCON instruction while in problem state.

- A specification interruption occurs if:
  1. Executing CE is in state 1 or 2.
  2. Executing CE's SCON bit (in its own CCR) is off.
  3. R1 specifies an odd GPR.
  4. Configuration mask specifies more than one CE, and an IOCE is selected.

- Condition code is set to:
  0 if all elements responded.
  2 if one or more elements failed to respond.

- Refer to Diagram 5-810 FEMDM.

The SCON instruction (Diagram 5-810, FEMDM) provides programmed control over the configuration of the 9020 system by setting up communication paths between its major elements. Basically, this instruction accesses general-purpose registers (GPRs) in the CE to obtain the selection and configuration masks previously set up by the program. The selection mask specifies which system elements are to be configured. The configuration mask specifies to each selected element: the state it is to assume, the computing element(s) from which it may accept future configuration changes, and the elements with which it may communicate. (Note the differences in the selection and configuration masks used by the 9020D and 9020E systems. The selection mask is one word long, and the configuration mask is a doubleword long. However, the second word of the configuration mask is used only by the 9020E system. The microprogram for the SCON instruction is not affected by the differences in the mask formats.)

At the start of execution, the selection mask is transferred to the select register, and the first word of the configuration mask is transferred to the external register. The instruction then checks for privileged operation and specification-type errors. A privileged operation error occurs if the CE attempts to execute the SCON instruction while in the problem state. A specification error occurs if:

1. The R1 field of the SCON instruction specifies an odd GPR (i.e., the configuration mask is not on a doubleword boundary).
2. The CE executing the SCON instruction is in state 1 or 2.
3. The CE executing the SCON instruction does not have its own SCON bit set in its configuration register.
4. The scon field of the configuration mask is equal to 0 (i.e., no valid CEs are specified by the SCON instruction).

If the selection mask specifies one or more IOCEs, the SCON instruction checks that not more than one CE communication bit is set in the configuration mask. Failure to meet this requirement results in a specification error and end-op.

The SCON instruction must activate a 5-usec 'reconfigure select' signal to the elements specified in the selection mask. Upon receipt of this signal, each selected element checks its own CCR to establish that the SCON bit for the issuing CE has been set. If so, the selected element ingates the configuration mask information from the issuing CE into its CCR. If the SCON bit is not set, the element will not accept the configuration mask. In addition, within a 9020E system, the DEs will accept the configuration mask only if (1) no more than four display generator communication bits are set, (2) no display generator has more than one bit set, and (3) no more than one display generator is configured to any one of the data registers (i.e., A, B, C, D). Also, within a 9020E system, the reconfiguration control unit will accept the configuration mask only if one or no IOCE communication bit is set.

To obtain a 5-usec select timeout for the 'reconfiguration select' signal, the SCON instruction sets up a constant of 19 (hex) into the B register and decrements B by 1 on every machine cycle. After the drop of the 'reconfigure select' signal, the issuing CE waits for 5 usec for the selected elements to respond. To obtain the 5-usec timout, the SCON instruction sets up a constant of 19 (hex) in B, and then decrements B by 1 on every machine cycle. (To obtain a constant of 19 in B, the microprogram first sets a constant of 0C in B and D. D is then added to B, and the result, incremented by 1, is transferred to B.)

The CE gates the configuration mask to the SEs and DEs via the SDBI. Consequently, the configuration mask is transferred to S and T. The second word of the configuration mask is used by the DEs only. This word is transferred from the odd GPR (specified by R1+1) to the T register. Because the CE communication field for SEs and DEs is ingated from SDBI(56–59), this field must be moved to the same bit positions of the T register.

The drop of the 'reconfigure select' signal causes the selected elements to respond. However, the selected elements respond only if they received correct parity for the configuration mask; if the element detects incorrect parity, it does not respond to the issuing CE.

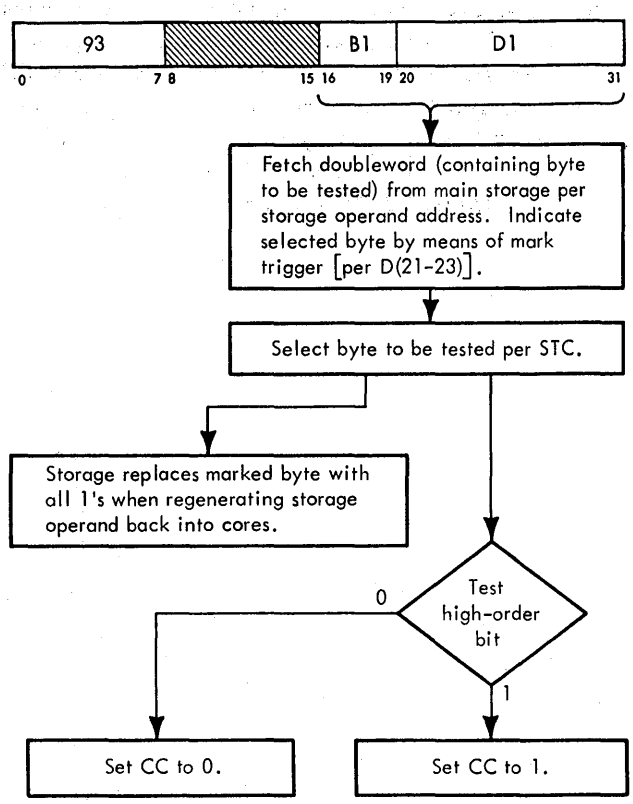
The response signal from each element is used by the issuing CE to reset a corresponding bit in the select register. Thus, at completion of the response timeout, the select register will contain all-0's, provided that all selected elements responded. If an element fails to respond, its corresponding bit in the select register is not reset.

The contents of the select register are stored into the GPR specified by the R1 field. At this time, the microprogram also examines the contents of the select register to establish if all elements responded. If all the elements did respond, the condition code is set to 0; if one or more elements failed to respond, the condition code is set to 2.

TEST AND SET, TS (93)

- Test high-order bit (bit 0) of storage operand byte (in storage), set CC according to state of tested bit, and set addressed byte back into storage as all 1's.

- SI Format:



- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.

- CC setting:
  High-order bit = 0: CC = 0.
  High-order bit = 1: CC = 1.

The Test and Set (TS) instruction tests the high-order bit of a single byte in main storage and then sets the byte tested to all-1's. The byte to be tested and set to 1's is specified in the storage operand address. The result of the test of the high-order bit is recorded in the CC: if the high-order bit is a 0, the CC is set to 0, if a 1, the CC is set to 1.

Diagram 5-811, FEMDM, is a flowchart of the execution of the Test and Set instruction. The first 16 bits of the instruction are in E, and the storage operand address is in

D. The immediate operand and the storage operand requested during I-Fetch are not used. Instead, a storage request per D is made during execution accompanied by a 'test and set' signal. This action allows no other access to this storage location between the fetching and regeneration of the byte. Before issuing the storage request, a mark trigger is set per D(21–23), via the STC, for use later by the storage unit.

The storage unit performs a unique regeneration operation for the Test and Set instruction. The addressed storage doubleword is fetched and set, unaltered, onto the SDBO, exactly as during a fetch operation. Unlike a normal fetch operation, however, the storage unit uses the mark bit supplied by the CE to designate the byte to be changed. When regenerating the 72-bit word, the storage unit sets the designated byte in core storage to all-1's. Thus, the storage unit does a combination store and fetch operation. The storage protection facility operates as normal for a store operation.

While generating the request for the storage operand, an address store compare test is started, similar to the test performed during the SS I-Fetch operation. Its purpose is to detect if the storage operand is buffered in Q. If it is, the 'PSC' trigger is set, forcing an exceptional condition during the next I-Fetch, thus refilling Q.

The bit to be tested must be placed into T(32). Therefore, the second operand is gated from the SDBO to AB, where the proper byte is selected and transferred to T(32–29) via the serial adder and F. An early end op is taken, overlapping the setting of the CC.

SECTION 10. DISPLAY INSTRUCTIONS

This section discusses the three instructions (RPSB, CSS, and CVWL) designed to perform the tasks of filtering and reformatting input radar, single-symbol, and weather-line data, and of assembling and updating images for the PVDs connected to the 9020E system. A fourth instruction, Load Chain (which provides for accessing chained or linked blocks of data), is also described. Before the instructions are discussed, however, the following paragraphs are presented as an introduction to the jobs to be performed and the methods used to accomplish them.

INTRODUCTION TO DISPLAY INSTRUCTIONS

The main purpose of these instructions is to provide an up-to-date display image for each of the PVDs attached to the 9020E system. One PVD's display image is an area located within the boundaries of its DE's core storage, which is variable in length and is designated 'refresh memory'. The display image is updated by assembling, under control of the Repack Symbols instruction (RPSB), a new image in another location in the DE's core storage and, when this image is complete, causing the display data for the PVD to be fetched from there. At the time updating of the display image begins, the area then being used to supply the PVD with display data is referred to as 'old refresh memory', and the updated display image to be constructed is referred to as 'new refresh memory'.

Software supplies the information that controls the execution of Convert and Sort Symbols (CSS), Convert Weather Lines (CVWL), and Repack Symbols (RPSB). This control information is communicated to the instructions by placing it in specified general-purpose and floating-point registers. The contents of the GPRs and FPRs are shown in the detailed write-ups of the respective instructions, along with an explanation of the use of the information contained in each register. Software also assembles and formats two groups of input data called input data streams, which reside in an SE and which CSS and CVWL use as input.

CSS selects from its input data stream all of the data pertaining to the PVD being updated converts it to the scale of the PVD, and sorts it into 16 storage areas (in an SE) assigned to the PVD, called sort bins. In subsequent execution, RPSB updates one-sixteenth of a PVD's viewing area (i.e., one sort bin) by incorporating this new data into the PVD's new refresh memory along with older data retained from the PVD's old refresh memory.

CVWL selects from its input stream all of the weather lines pertaining to the PVD being updated. The instruction converts the selected weather lines to the scale of the PVD and stores them directly into the PVD's new refresh memory.

One execution of CSS or CVWL processes data for a complete PVD. One RPSB execution processes only one-sixteenth of the data (one sort bin) for one PVD and must be issued 16 times to update a complete PVD.

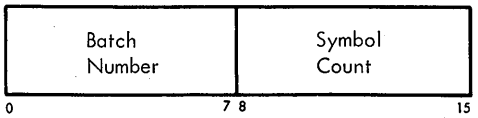
Because RPSB differs greatly from CSS and CVWL, which are similar in operation, the discussion is now divided into two sections.

Introduction to RPSB

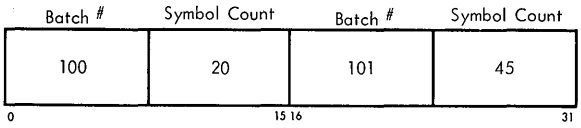
RPSB is a complex instruction, and a knowledge of many unique names and terms is needed for an understanding of the operations performed during its execution. There are two classes of data on which RPSB operates: radar data and single-symbol data. Radar data consists of both history and current symbols, which are displayed on the PVD. Single-symbol data consists of current symbols only. RPSB execution is identical for both classes, except that all operations dealing with history data are deleted when operating on single-symbol data. For this reason, the processing of radar class data is discussed here; the terms are defined as they are encountered.

Information displayed on the face of the PVD is of two basic types: current information and history information. An example of current data, as seen on the face of the PVD, is a symbol depicting the present position of an aircraft passing through the PVD's area. An example of history data is the symbol representing the position of the aircraft some seconds before it reached its present position. History data is displayed at half brightness, as compared with current data, and several history targets are displayed with each current target. They form a trailing line showing the path of the aircraft.

A PVD fetches the information it will display from its refresh memory area in the DE's core storage. This area is described by a table called a descriptor table. It is made up of halfword descriptors, each containing a batch number and a symbol count.



The batch number, determined and assigned by software, identifies the time period and radar sector from which that particular group of symbols originated. The symbol count indicates the number of symbols contained in the batch.



The two descriptors shown indicate that the first 20 symbols in refresh memory are from batch number 100 and that the following 45 symbols are from batch 101. The descriptors are taken in sequence as illustrated; i.e., the first descriptor in the table describes the first group of symbols in refresh memory, the second descriptor describes the second group, etc. At the beginning of execution the refresh memory area is designated old refresh memory (ORM), and the table describing it is called the old descriptor table (ODT). RPSB execution generates a new descriptor table (NDT) from the ODT and, then, by using the symbol count accumulated when building the NDT, constructs a new refresh memory area (NRM).

A work control table (WCT), furnished by software, provides the necessary control for the updating of refresh memory. WCT orders determine the sequence and the functions to be performed. Seven WCT orders are provided:

Delete N1
Delete N2
Delete
Modify
Insert
EOCT
EOB

A typical WCT is shown in the detailed write-up of RPSB, along with a discussion of each of the WCT orders. These orders may appear in various combinations in the WCT. The sequence used later in this text, using most of the orders, is intended only as an example. The following sequence of orders is also possible, as are many others:

Bin displacement table
Modify
Insert
Bin displacement table
Modify
Insert
Bin displacement table
Insert
Bin displacement table
EOCT

} Typical WCT order sequence to insert a radar class/type.

Note that a delete order is not necessarily the first order.

In addition to the WCT orders, the RPSB microprogram provides for moving, first, history descriptors from the ODT to the NDT and, later, current descriptors from the ODT to the NDT. If no delete orders are in the WCT, RPSB immediately moves history descriptors to the NDT. Delete orders (could be one or more) present in the WCT, however, indicate that one or more of the history descriptors in the ODT are to be deleted (those descriptors that describe symbols that no longer need to be displayed). One (or more) of the three delete orders deletes the proper descriptors from the ODT and, then, updates the ORM starting address to bypass those symbols so that they will not be moved to NRM. After this has been accomplished (indicated by no further delete orders), the move history descriptors routine is entered. Its function is to accumulate a history count and to move the history descriptors (which were not deleted) to another location in storage, the NDT. Once in the NDT, they eventually cause RPSB to move the history symbols that are still to be displayed from ORM to NRM. Descriptors are moved until a null descriptor (all 0's) is encountered. This descriptor separates history descriptors from current descriptors in the ODT.

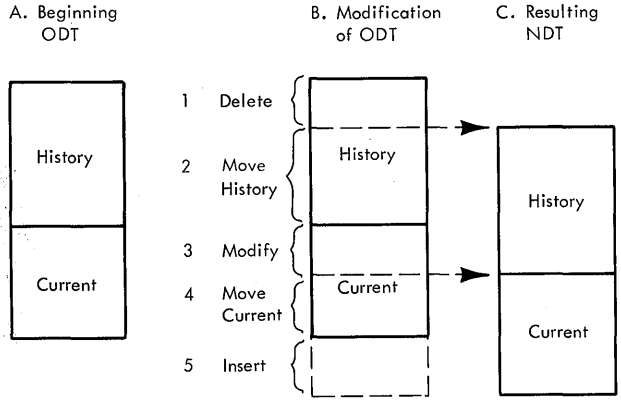
RPSB then examines the next order in the WCT. If the order is a Modify, it indicates two things: (1) some of the current symbols now in ORM, and described by some of the current descriptors in the ODT, are no longer current and must be modified; (2) since the WCT contains no Modify orders when the input class type is single-symbol data, it indicates that the input class type being processed is radar data. The primary function of this order (Modify) is to move current ODT descriptors (describing those symbols to be modified) to NDT positions adjacent to the last history descriptor. Later (in the execution), this will cause the symbols to be modified (reset the brightness bit) as they are moved from ORM to NRM. One current descriptor per Modify order is moved (if the batch number in the Modify order matches the batch number in the descriptor) from the ODT to the NDT, and the symbol count for each descriptor moved is added to the accumulated history count. After the last Modify order has been executed, a null descriptor is inserted in the NDT to separate the history and current descriptors.

RPSB then enters a routine whose objective is to move the remaining current descriptors from the ODT to the NDT. Descriptors are moved and a current count is accumulated until a second null descriptor is encountered in the ODT. This all-0's descriptor indicates the end of the ODT and causes RPSB to check the next WCT order.

Finding an Insert order in the WCT indicates to RPSB that new symbols are to be entered into new refresh

memory. For these symbols to be moved, a descriptor must be constructed and inserted into the NDT following the last current descriptor stored there. The insert order itself provides the batch number, and, because the new symbols are located in a sort bin (as the result of a CSS execution), RPSB calculates the symbol count for the descriptor by subtracting the starting address of the sort bin from the sort bin stop address. (These addresses are furnished by software as part of the control information.) The newly assembled descriptor is then stored in the NDT. More than one Insert order can be present in the WCT; if there are more, RPSB repeats the previous operation and uses the symbol counts calculated for the descriptors to accumulate a total insert count.

After all insert descriptor orders have been executed, a null is inserted in the NDT, and the new descriptor table (NDT) is completely built, containing two kinds of descriptors (history and current), as did ODT.



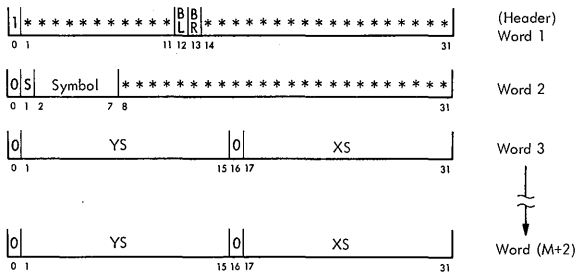
As the descriptors are moved into the NDT, their symbol counts are used to accumulate a total history count, a total current count, and a total insert count. In the following routines, which move the actual display words (referred to as symbols), these counts control the movement of the three different types of symbols. The counts are decremented as the symbols are moved. When both the history and current counts have been decremented to 0, all the pertinent symbols have been moved from ORM to NRM. The new symbols, located in a sort bin (rather than ORM), are now moved to NRM under control of the insert count. When this count reaches 0, one complete class type has been processed, and the next WCT order must be an End of Class Type (EOCT) order. An End of Block (EOB) order following the EOCT in the WCT indicates that RPSB has completed processing data for one bin of one PVD. Any other WCT order following the EOCT order indicates that there is another class type of data to process, and the microprogram branches back to the beginning routines.

Introduction to CSS and CVWL

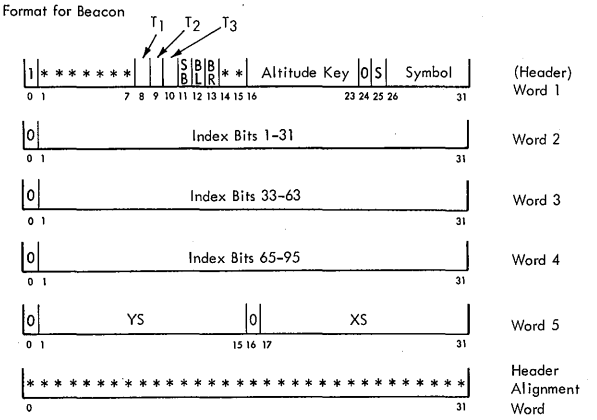
CSS and CVWL are similar in that each selects its input from a similar stream of data containing coordinates that will eventually be displayed on many different PVDs. The PVD on which each target or weather line will be displayed is determined by each coordinate's respective geographic location. One execution of either instruction selects (from its input data stream) all of the input data for one PVD. Both instructions must be reissued to process their input data streams for another PVD.

The CSS instruction processes an input stream of primary radar/single-symbol data or beacon data. Note that beacon data is also radar data. Formats for these two basic types of data are as follows:

Input Format for Primary Radar/Single Symbols.



Input Format for Beacon



Legend:

BL Blink
BR Brightness
S Symbol Size
T_1, T_2, T_3 Type bits
SB Selected Beacon
M Number of X, Y position words in primary radar/single-symbol data block (if necessary, a dummy position word of all zeros must be included to make the count even)

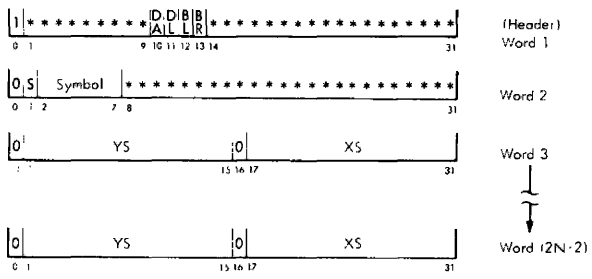
* Ignored by CSS

Note:
Multiple data blocks accepted for input.

CSS selects (from the input data stream) all the data that is valid for one PVD and stores it, according to the targets location on the face of the PVD, in the appropriate sort bin (there are 16). The data, stored in the sort bins, is in word format and contains all the information required to display the target.

The CVWL instruction processes an input data stream of weather-line coordinates destined for many different PVDs. Format of this input data stream is as follows:

Input Format



Legend:

DA	Dash
DL	Dash length
BL	Blink
BR	Brightness
S	Symbol Size
N	Number of weather lines
*	Ignored by CVWL

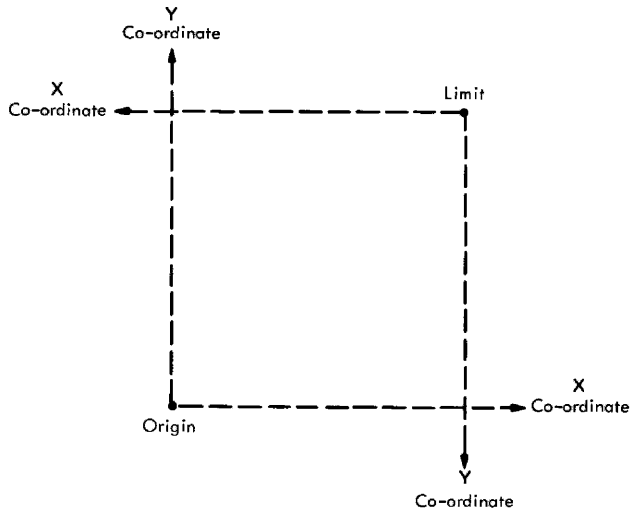
It selects all of the weather lines (pairs of coordinates) that are valid for that PVD and stores them in the PVDs refresh memory as a doubleword containing all information required to display the weather line.

All control information for CSS and CVWL is located in GPRs and FPRs. These registers must be set up properly by software prior to issue of each CSS or CVWL instruction. The detailed discussion of each instruction presents the format of the GPRs and FPRs and explains the use of their contents.

Control information needed for execution of the CSS instruction consists of the geographical boundaries of the PVD, the geographical boundaries of each of three sterile areas located within a PVD's geographical boundary, the sort-bin base address and the displacement values for the 16 sort bins, the PVD index (for beacon input), the altitude mask (for beacon input), the type bits (for beacon input), the address in prime storage from which to read the next input doubleword, the input data count (indicating the number of words or beacon data blocks in the input stream), and the PVD's conversion constant.

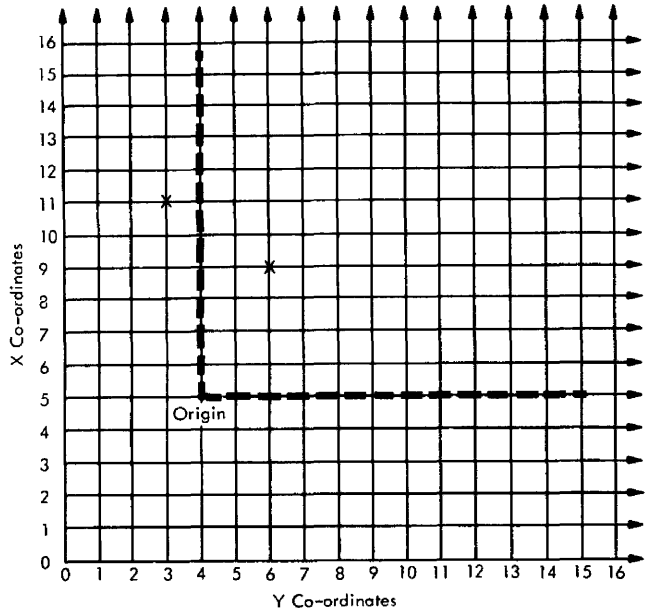
Control information needed for execution of the CVWL instruction consists of the geographical boundaries of the PVD, the nine-tenths border region coordinates of the PVD, the geographical boundaries of each of three sterile areas located within a PVD's geographical boundary, the address in prime storage from which to read the next input doubleword, the address in display storage for storing the next output doubleword, the input count indicating the number of doublewords in the input stream, and the PVD's conversion constant.

CSS and CVWL perform geographic and sterile area filtering. The geographic filter determines whether the input target (in the case of CSS) or either end of the weather line (in the case of CVWL) is within the geographic area of this PVD. Two sets of coordinates, which describe the geographic area of the PVD, are supplied by software and are in FPR 0. One set defines a point called the geographic origin of the PVD; the other set defines a point called the geographic limit of the PVD. The origin is the lowest-left point of the area and the limit is the uppermost right point. Using these two points and following the coordinates, a square can be drawn which is the geographical area.



The coordinates of the origin are subtracted from the coordinates of the input point. A point within the PVD's area will have coordinates of a higher value than those of the origin. If both X and Y subtraction results are positive,

the input point could be within the area. Either result negative means that the point cannot be in the PVD's area.



Example One

This point is within the PVD area-
both results are positive.

Input Co-ordinates		Y6	X9
Origin Co-ordinates	-	Y4	X5
Result		+Y2	+X4

Example Two

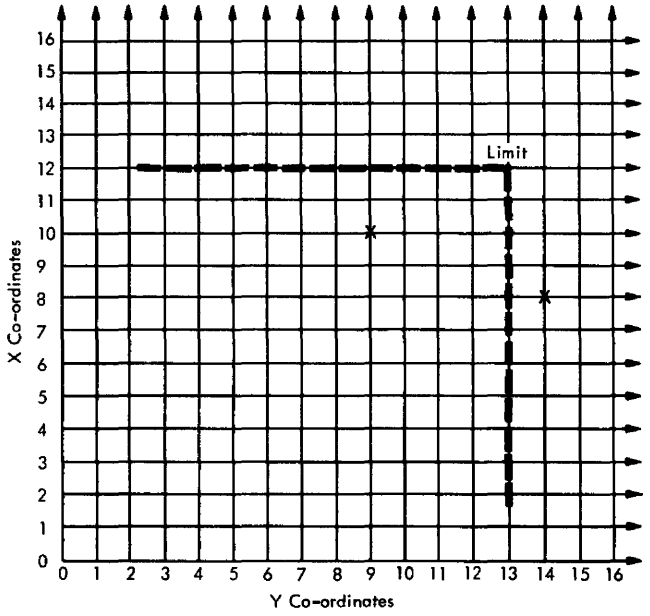
This point is not within the PVD area-
the Y result is negative.

Input Co-ordinates		Y3	X11
Origin Co-ordinates	-	Y4	X5
Result		-Y1	+X6

A latch testable by the microprogram (the 'limit' latch) is used to indicate the results of all of the filters. According to the nature of the filter being performed, this latch is set or reset per the results of the subtractions. If both X and Y results are not positive, the 'limit' latch is set now to indicate that the input point failed the geographic filter.

In CSS and CVWL, the origin is allowed to be negative. When the quantity involved in the subtraction is negative, no test is made on that subtraction (X or Y or both). In this case the maximum negative origin cannot be allowed to exceed the PVD scale.

The second part of the geographic filter is performed exactly as the first, but, this time, the coordinates of the geographic limit are subtracted from the input coordinates. Negative X and Y subtraction results indicate that the input



Example One

This point is within the PVD area-
both results are negative.

Input Co-ordinates		Y9	X10
Limit Co-ordinates	-	Y13	X12
Result		-Y4	-X2

Example Two

This point is not within the PVD area-
the Y result is positive.

Input Co-ordinates		Y14	X8
Limit Co-ordinates	-	Y13	X12
Result		+Y1	-X4

point could be within the PVD's area. This time, the 'limit' latch is set if both results are not negative.

The geographic filter is now complete, and the microprogram tests the 'limit' latch. The 'limit' latch on indicates that the input target (CSS) or one end of the weather line (CVWL) has failed the filter. If the 'limit' latch is off after the geographic filter, the input point is within the PVD's geographic area.

The purpose of the sterile area filters is to determine whether the input target (CSS) or one end of the weather line (CVWL) is within the boundary of any of the three sterile areas. Each of the sterile areas is also defined by two

sets of coordinates. The sterile-area filtering can be thought of as performing geographic filtering in reverse; i.e., the filter will be passed if the target does not fall within the geographic area defined by the origin and limit coordinates. The actual operations performed are duplicates of those within the geographic filters, the only differences being the coordinates describing the sterile areas and the interpretation of the setting of the 'limit' latch. The sterile area coordinates, furnished by software, are obtained by CSS and CVWL from FPRs (sterile area 1 from FPR-2, sterile area 2 from FPR-4, and sterile area 3 from FPR-6). A target point or a line with either end point in any one of the sterile areas will be rejected, and the instruction will start processing the next input target (CSS) or weather line (CVWL).

Each PVD could have a different scale, depending on the size of the area it displays. For this reason, a separate conversion constant for each PVD must be furnished by software. The purpose of the conversion routine is to convert the input coordinates from system scale to PVD scale, using the conversion constant. This routine is common to both CSS and CVWL. Another routine common to CSS and CVWL is the time clock update routine. The coordinates to be converted are in B, and the conversion constant is in S, no matter where entry is from. B is multiplied by S, using the 'SEL-MPL * E3' micro-order, and the result is a 10-bit Y coordinate and a 10-bit X coordinate. Upon completion of the routine, STAT D on returns control to CVWL.

REPACK SYMBOLS, RPSB (0F)

- Assembles an updated display image (refresh memory) for one-sixteenth of a PVD's area.

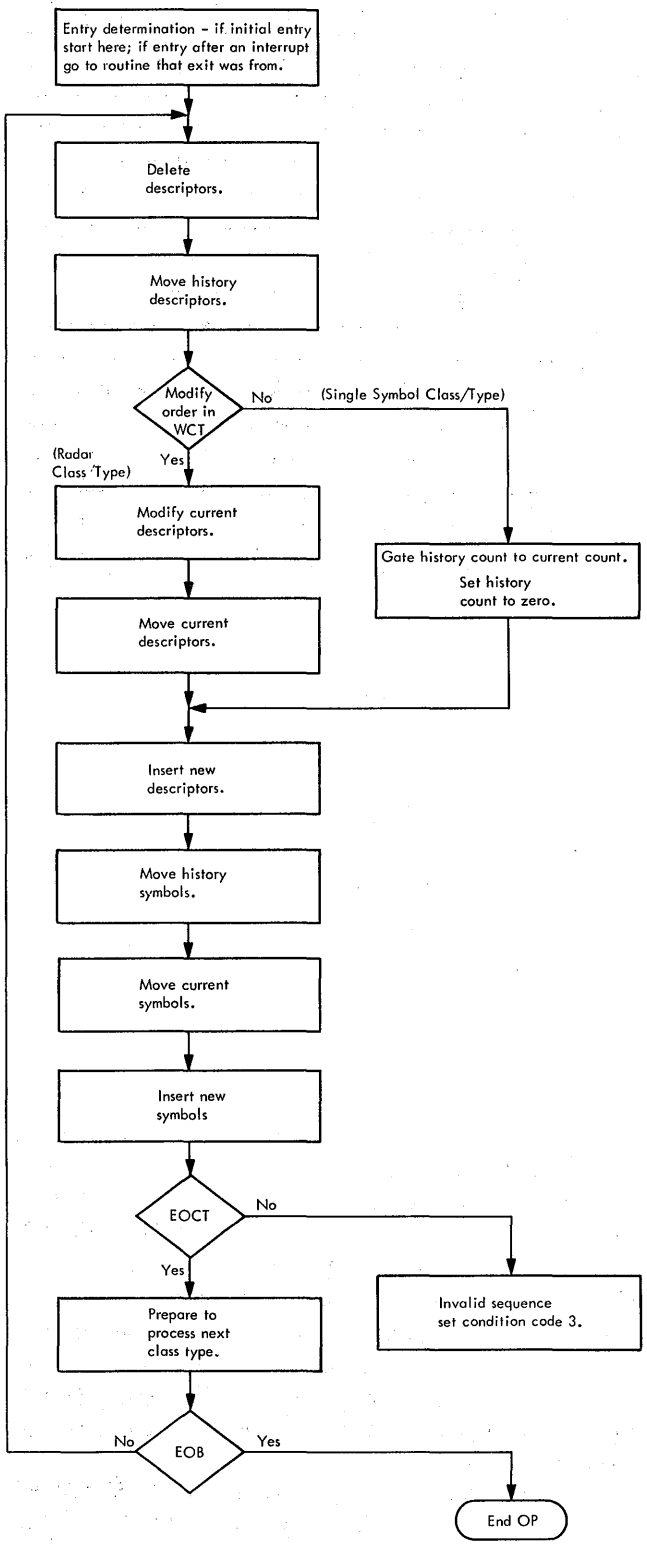
- Logically divided into two sections: (1) the assembling of a new descriptor table by moving descriptors, and (2) the assembling of a new display image by moving symbols (display words).

- Control information contained in GPRs, FPRs, WCT, and ODT.

- Program Interruptions:
  1. Protection (store or fetch protect violation)
  2. Addressing (input or output addresses outside available storage)
  3. Specification:
     a. New page address, contained in bytes 5–7 of doubleword following a 512-byte page, is not on a doubleword boundary.
     b. The Next Old Refresh Memory Addr (ORMA) in GPR 4 or the Next New Refresh Memory Addr (NRMA) in GPR 5 is not initially on a doubleword boundary.

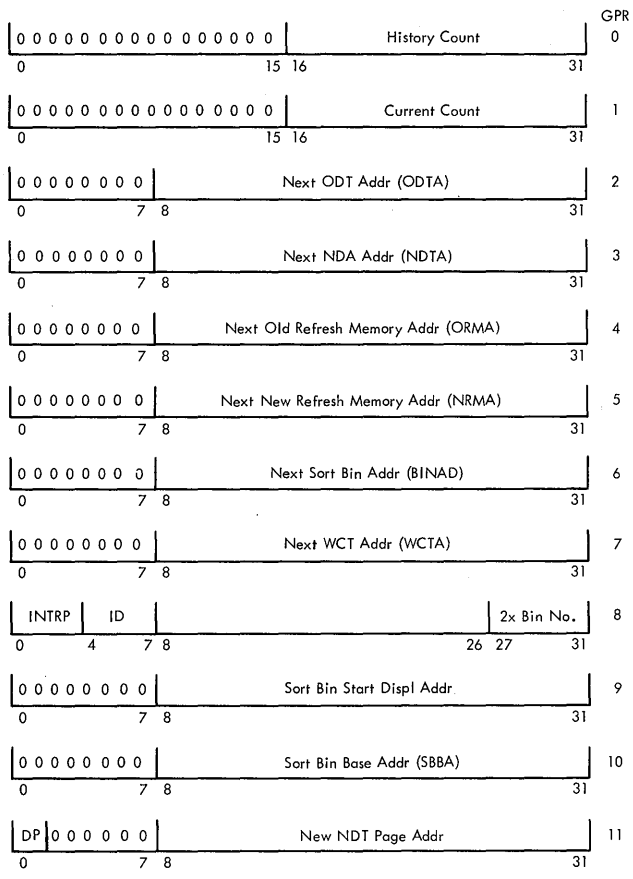
- Condition Code
  0–Processing complete
  1–Not used
  2–Page boundary encountered in the NDT
  3–Indicates an illegal sequence in the WCT orders

- RR Format



For a detailed discussion, read the following text and refer to Diagram 5–902, FEMDM.

RPSB is divided into two sections. The first section operates only with descriptors. It deletes some descriptors from the ODT, moves descriptors from the ODT to the NDT, and constructs new descriptors for the input data being processed. The second section processes only display words (symbols). It moves symbols from ORM to NRM and, then, from a sort bin to NRM. The control information required by RPSB to perform these tasks is furnished by software in the form of data stored in the GPRs and a work control table (WCT). Specific register contents are shown below, followed by a description of the use of each.



The function of each of the general registers 0–11 and floating-point registers 0–6 is as follows:

History Count–(GPR 0): Bits 16–31 are reserved for use by RPSB in accumulating a symbol count of history data. RPSB will not restore the original contents of GPR 0 at the end of its execution.

Current Count–(GPR 1): Bits 16–31 are reserved for use by RPSB in accumulating a symbol count of current data. RPSB will not restore the original contents of GPR 1 at the end of its execution.

Next ODT Addr (ODTA)–(GPR 2): Bits 8–31 maintain the address of the next halfword descriptor in the old descriptor table. It is updated on a two-byte basis. When the last descriptor has been processed from the previous doubleword, a new doubleword is fetched from the location indicated by this address (on a doubleword boundary).

Next NDT Addr (NDTA)–(GPR 3): Bits 8–31 maintain the address of the next halfword descriptor in the new descriptor table. It is updated on a two-byte basis.

Next Old Refresh Memory Addr (ORMA)–(GPR 4): Bits 8–31 maintain the current word address of the next symbol in old refresh and are updated on a doubleword basis (two symbols per doubleword). A specification exception occurs if this address is not initially on a doubleword boundary.

Next New Refresh Memory Addr (NRMA)–(GPR 5): Bits 8–31 maintain the current word address of the next symbol in new refresh and are updated on a doubleword basis (two symbols per doubleword). A specification exception occurs if this address is not initially on a doubleword boundary.

Next Sort Bin Addr–(GPR 6): Bits 8–31 are reserved for use by RPPSB for holding the sort bin address from which the next symbol for the current class/type may be fetched. RPSB calculates the initial address from the sum of Sort Bin Base Addr (GPR 10) and the appropriate Bin N Disp Addr value from the bin displacement value table at the beginning of the work control table. This address is not used or modified until RPSB begins to fetch symbols from the sort bin, reformat them, and insert them into new refresh. At this time, it is updated on a doubleword basis (two symbols per doubleword). The referenced sort bin data is a result of a previous CSS operation. This address points to the proper location for the next class type at the end of insertion of all new symbols. The original contents of GPR 6 will not be preserved during RPSB execution.

Next WCT Addr (WCTA)–(GPR 7): Bits 8–31 contain the next address in the work control table. The address is always on a halfword boundary.

INTRP ID–(GPR 8): The INTRP ID (interrupt identification code) is reserved for use by RPSB and is normally set to 0 to indicate that instruction execution is to start from the beginning. When RPSB detects a pending interrupt condition (I/O or External) at one of the several break points in the instruction, it sets the associated INTRP ID code and adjusts the instruction address field of the current PSW so that it points to the RPSB instruction itself. Then it terminates to allow the interruption to take place.

The INTRP ID code is also set whenever a time clock step request is detected. However, in this case, the time clock stepping is accomplished within RPSB itself. Upon stepping of the time clock, RPSB tests for an interrupt condition. If an interrupt is pending, the action described above is performed and the instruction is terminated; if no

interrupt request was made, RPSB re-initializes and re-enters the instruction at the point from which the time clock step was acknowledged (without termination).

Bits 0–3 specify whether the interrupt code pertains to descriptor table updating or to display image updating. These codes are as follows:

Bits 0–3

0000	Descriptor table updating - no odd delete adjustment is required on the first valid descriptor to be moved after the delete operations.
0001	Descriptor table update - the symbol count of the first valid descriptor to be moved (for the current class/type) after descriptor deletions will be increased by one.
1001	Display image updating - all codes with 1001 for bits 0–3 pertain to symbol moving from the old refresh area to the new refresh area (DE to DE) or to the insertion of new symbols into the new refresh area (SE to DE).

Bits 4–7 identify specific interrupt subcategories of descriptor table or display image updating and are defined below:

For descriptor table updating (bits 0–3 = 0000 or 0001), bits 4–7 are defined as follows:

0001	Exit at end of delete routines. Re-entry at beginning of move history descriptors routine.
0010	Exit at end of move history descriptors routine. Re-entry at beginning of move and modify current radar descriptors routine.
0100	Exit at end of move and modify current radar descriptors routine. Re-entry at beginning of move current descriptors routine.
0110	Exit at end of move current descriptors routine. Re-entry at beginning of insert new descriptors routine.

All of the following codes will cause re-entry into the instruction at the same point, i.e., at the beginning of the move history current symbols routine. Program interrupts will be caused by either an invalid DE address or a DE storage protection exceptional condition.

For display image updating (bits 0–3 = 1001), bits 4–7 are defined as follows:

0001	Nonprogram interrupt exit from end of insert descriptors routine or from within the move history symbols loop.
0010	Program interrupt exit from within the move history symbols loop.
0011	Nonprogram interrupt exit from within the move current symbols loop.
0100	Program interrupt exit from within the move current symbols loop.
0101	Nonprogram interrupt exit from end of move history and current symbols routine.
0110	Program interrupt exit from end of move history and current symbols routine.
0111	Nonprogram interrupt exit from within the insert new symbols loop when NRMA (GPR 5) and BINAD (GPR 6) are on alternate word boundaries.
1000	Program interrupt exit from within the insert new symbols loop when NRMA (GPR 5) and BINAD (GPR 6) are on alternate word boundaries.
1001	Nonprogram interrupt exit from within the insert new symbols loop when NRMA (GPR 5) and BINAD (GPR 6) are on the same word boundaries.
1010	Program interrupt exit from within the insert new symbols loop when NRMA (GPR 5) and BINAD (GPR 6) are on the same word boundaries.
1011	Nonprogram interrupt exit after last new refresh memory store.
1100	Program interrupt exit after last new refresh memory store.
1101	Nonprogram interrupt exit before last new refresh memory store.
1110	Program interrupt exit before last new refresh memory store.

In the event of a nonprogram interrupt, all addresses and symbol counts are adjusted to reflect the symbols actually moved to the new refresh area. For a program interrupt, however, residual symbol counts and data addresses vary depending upon the particular interrupt condition.

Bin No.–(GPR 8): Bits 27–31 contain a number which is twice the sort bin number (0–15) for the sort bin currently being processed. It is used for two main functions: (1) to enable RPSB to access the proper bin displacement value in any bin displacement value table in the WCT, and (2) to indicate the number of the sort bin processed by RPSB to the operational program in the event of any termination of the instruction.

Sort Bin Start Displ Addr–(GPR 9): This register is reserved for use by RPSB. For any particular Insert order in process during the insert operation, bits 8–31 of GPR 9 hold a start displacement value for sort bin data. It is used for calculating a symbol count for the given Insert. For the first Insert order in the first class/type processed, this displacement value is taken from the initial bin displacement value table in the WCT. For each succeeding

Insert order (for every Insert order in all class/types processed in any RPSB execution), this displacement is the stop displacement value for the previous Insert order. (The stop displacement value for any Insert order is taken from the associated bin displacement value table.) The original contents of this register will not be preserved during instruction execution.

Sort Bin Base Addr (SBBA)–(GPR 10): Bits 8–31 contain the sort bin base address used in calculating the BINAD (GPR 6) from the sort bin displacement values in the WCT. It must always be on an N512–8 page boundary (i.e., the low-order 9 bits must contain the hex value 8).

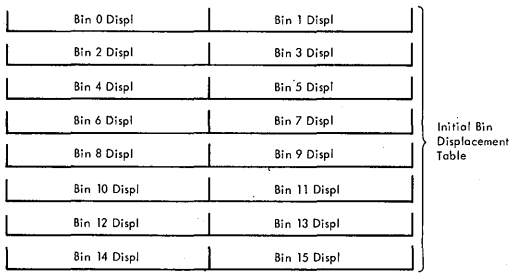
DP–(GPR 11): The DP (descriptor paging code) bits 0 and 1 are reserved for use by RPSB and are normally set to 0 to indicate that instruction execution is to start from the beginning. Whenever RPSB encounters an NDT page overflow (63 doublewords of new descriptors have been stored in an NDT page), it sets the DP bits to 1, inserts the New NDT Page Addr (GPR 11) in the page link field of the old page, and replaces the Next NDT Addr in GPR 3 with a new page address from GPR 11. Later, when RPSB reaches a convenient break point in its execution (end of given class/type), the DP bits, being on, will cause RPSB to terminate the instruction with CC2.

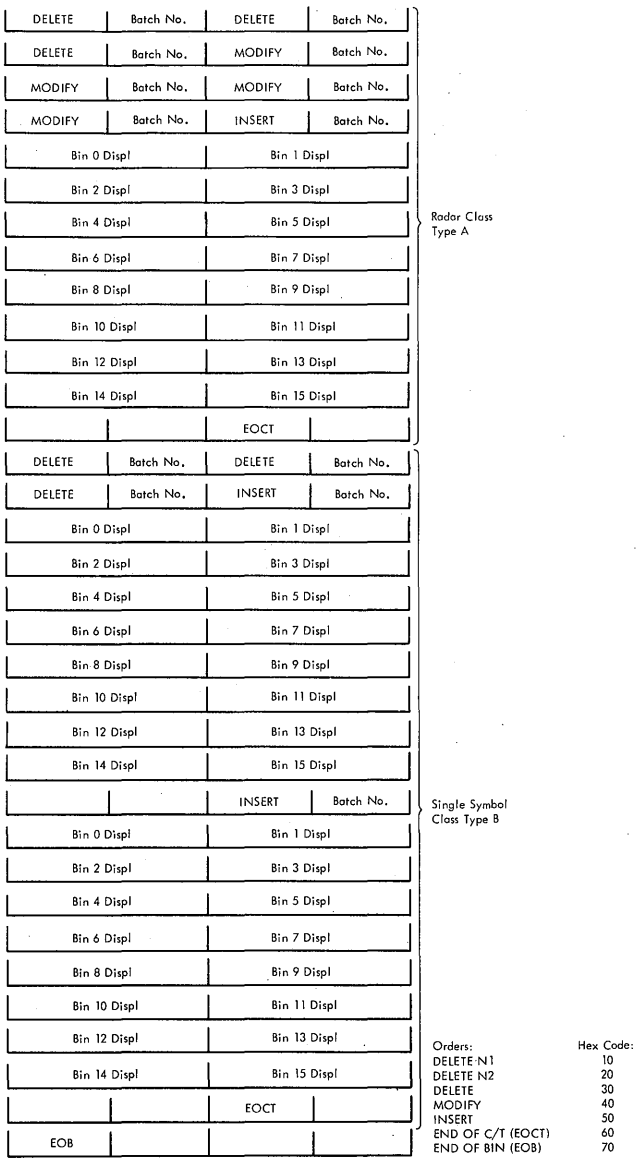
New NDT Page Addr (NDPA)–(GPR 11): Bits 8–31 contain an address of a new page in the new descriptor table, to be used for replacement of Next NDT Addr in GPR 3 in the event of an NDT page overflow.

GPR's 12–15: These registers may be used by the programmer.

FPR 0, 2, 4, 6: These registers are reserved for use by RPSB as working registers. The original contents of these registers will not be preserved during instruction execution.

The work control table contains information on bin displacement values and various control orders. It is applicable to all 16 sort bins, although RPSB operates on only one sort bin per execution. A typical WCT is shown below, followed by a description of each entry (bin displacements and orders).





At the beginning of the WCT, and immediately following each Insert order, is a table of 16 bin displacement values. The table must always start on a word boundary. The bin displacement values are defined as follows:

Bin N DISPL: These are halfword operands, each of which is a displacement value. They are used in two ways: (1) to calculate BINAD (GPR 6) upon initial entry into the instruction (using the initial bin displacement value table), and (2) as stop and start displacements for determining a symbol count for each Insert order. When generating the symbol count for a new descriptor for a given Insert order, the stop displacement is taken from the bin displacement

value table associated with the given Insert order, and the start displacement is taken from SORT BIN START DISPL ADDR in GPR 9 (see same).

Since these displacement values are assigned the same values used in CSS, they are subject to the same restrictions (see CSS instruction). Each initial displacement value will have been assigned any value lying on any word boundary, except for the following:

1. 0 to 27 (hex)
2. FE01 to FFFF
3. 1FC or 1F8 (hex) in the low-order 9 bits
4. Only one sort bin to each displacement value

Each WCT order consists of a halfword: the first byte contains the order code; the second byte contains a batch number to be compared against the batch number contained in ODT halfword descriptors. (Batch numbers of 0 are illegal.) WCT orders are as follows:

Delete N2: This order deletes all ODT descriptors up to, and including, the first null (all -0's) descriptor. Only one Delete N2 order may be issued for a class type.

Delete N1: This order deletes all ODT descriptors up to, but not including, the first null descriptor. Only one Delete N1 order may be issued for a class type.

Delete: If the batch number of the current descriptor matches the batch number of this order, this order instructs RPSB to delete the number of symbols specified by the descriptor from old refresh and to delete the given descriptor. Otherwise, RPSB proceeds to the next order in the WCT. If the batch number of this order is 255, it is a dummy Delete order. (The batch number comparison will always be unsuccessful since no batch of data will receive a number as high as 255). It will be used only to align, if necessary, the first Insert order on an odd halfword boundary.

Modify: If the batch number of the current descriptor matches the batch number of this order, RPSB moves the current descriptor from the ODT to the NDT. In addition, if there is a match, RPSB will modify (reset the brightness bit) the old refresh data associated with that batch and move the data to new refresh memory. If the batch numbers do not match, RPSB proceeds to the next order in the WCT. If the batch number of this order is 255, it is a Modify order (like the dummy DELETE). This order signifies to RPSB only that radar data is being inputted (Modify orders are never used with single-symbol data). In addition, for any radar class/type sequence of WCT orders, it will be used to align, if required, the first Insert order on an odd halfword boundary. A third use of a dummy MODIFY occurs when it is placed after an Insert order. Encountered in this position it instructs RPSB to insert a null descriptor in the next available location in the NDT. RPSB always steps the WCT address four bytes to the next WCT order after encountering this dummy Modify.

Insert: This order instructs RPSB to extract symbol data (previously processed by CSS) from the sort bin, to construct a new descriptor with the batch number (given in this order) and the count of symbols inserted, and to add the new descriptor to the NDT. The next order in the WCT to be accessed by RPSB will be taken from the eighteenth halfword position from the given Insert order. Every Insert order will lie on an odd halfword boundary (i.e., not a word boundary) and is immediately followed by its own table of 16 bin displacement values (always on a word boundary).

EOCT: This order (End of Class/Type) reinitializes RPSB for the next class/type pass. Then RPSB proceeds to the next WCT order.

EOB: This order (End of Bin) signifies the end of processing of the given sort bin and terminates the instruction. When used, it must always follow an EOCT order.

Control of refresh storage management via the Work Control Table and descriptor tables is illustrated, as follows, for single-symbol data and radar data:

Single Symbols: For each class/type of single-symbol data, the ODT contains some number of descriptors, each describing some number of symbols stored in old refresh. A null descriptor (all-0) in the ODT separates this class/type from the descriptors of the next class/type.

The WCT may contain orders for a single-symbol class/type as shown on the following examples:

1. Normal update cycle:
   0-n DELETE orders
   0-n INSERT orders
   1 EOCT order
2. Delete all data from
   a class/type:
   1 DELETE N1
   1 EOCT order
3. Insert data for a class/type:
   0 or 1 DELETE #255 order
   0-n INSERT orders
   1 EOCT order

The number of DELETE orders (0-n) must always be such that the first Insert order, if any, lies on an odd halfword boundary. One dummy Delete (batch 255) order may be required to ensure the proper alignment.

Radar: For each class/type of radar data, the ODT contains some number of history descriptors, a null descriptor (all-0), some number of current descriptors, and a final null descriptor (all-0). The first null descriptor

separates history from current while the second null descriptor separates this class/type from the next. Each of the non-zero descriptors defines some number of symbols in old refresh.

The WCT may contain orders for a radar class/type as shown in the following examples:

1. Normal update cycle:
   0-n DELETE orders
   1-n MODIFY orders
   0-n INSERT orders
   1 EOCT order
2. Delete data from a class/type:
   1 DELETE N2
   1 DELETE N1
   1 EOCT order
3. Insert data into a class/type:
   0-n INSERT orders
   1 MODIFY (255) order
   0-n INSERT orders
   1 EOCT order

A dummy MODIFY order may be required to align the Insert order on an odd halfword boundary. (Another dummy modify may precede this order to establish the class/type as a radar class/type.)

RPSB processing is terminated upon recognition of an EOB order following an EOCT order in the last class/type sequence.

Radar data is distinguished from single-symbol data by the presence of at least one Modify order. If no data is to to be modified, a dummy Modify (batch number 255) order must be used.

Radar data can also be treated as single-symbol data if no history data exists. In this case, the null descriptor separating history from current should be eliminated.

A description of the operational flow of the execution of the WCT orders is presented in 9020D and E System Principles of Operation.

Descriptors Section

At many points during RPSB execution, termination will occur if interrupts are pending. After a nonprogram interrupt, the instruction is reissued and is restarted at the point of interruption. For this reason, RPSB execution begins by determining whether entry to the instruction is initial entry or whether it is a re-entry after an interrupt. The INTRP ID code (GPR 8, 0–7) is tested, and, if bits 4–7 are equal to 0, this is an initial entry; if other than 0, this is a re-entry.

When re-entering RPSB after an interrupt, execution begins at the point of interruption. This is accomplished by testing the INTRP ID further. Bits 0–3 are checked first; if they are not equal to 9, the instruction was interrupted while operating in the descriptor section of the microprogram. In that case, the microprogram tests bits 4–7 to determine which routine in the descriptors section to re-enter (move history descriptors modify current descriptors, move current descriptors, or insert descriptors). Re-entry is to the beginning of the appropriate routine, provided that all control information in the GPRs and FPRs had been preserved throughout the interruption.

In the case where bits 0–3 of the INTRP ID are equal to 9, the instruction was interrupted while operating in the section that moves symbols to new refresh memory. This type of re-entry always starts with the routine that initializes for the move from ORM to NRM. This can be done because (before allowing the interrupt) the history, current, and insert-symbol counts are updated by RPSB according to the number of symbols that have been moved. For example, if 25 symbols have been moved and the beginning history and current counts were 10 and 20, respectively, the history count would be set to 0 and the current count to 5. The routine can now be entered as through it were initial entry and still can begin moving at the point of interruption.

Upon entry, if bits 4–7 of the INTRP ID are 0, this is not a re-entry after an interrupt. The descriptor paging (DP) bits are interrogated next. Set to 1's, they indicate that re-entry is after termination because of an NDT page overflow. This type of termination occurs only at the end of a class type; upon re-entry, RPSB determines whether there is another class type to process.

If bits 4–7 of INTRP ID are 0, and the DP bits are 0, this is an initial entry; the microprogram, using the address in GPR 7, fetches the first order from the work control table (WCT). The first order in the WCT may be one of the three types of delete descriptor orders. The first type of delete order checked for is the Delete N2, which deletes all ODT descriptors up to, and including, the first null (all 0's) descriptor. The second type of delete order, the Delete N1, deletes all ODT descriptors up to, but not including, the first null descriptor. These two delete orders share part of the same routine since the only difference between them is the point at which they stop deleting. To accomplish the deletes, the descriptors are read from the old descriptor table (ODT) and examined one at a time. If the descriptor is not a null, its symbol count is added to K, which, at the end of the routine, will contain the total number of symbols to be deleted. The next ODT Address (GPR 2) is updated by two; this effectively deletes the descriptor. (When the moving of descriptors begins, the first descriptor to be moved is the one addressed by the next ODT address.) The microprogram now branches back to examine the next ODT descriptor. This process continues until a null descriptor is encountered. For a Delete N2, the null descriptor is deleted by updating the ODT as before. This step is bypassed for a Delete N1 order. The WCT is updated

by two, to point to the next order; the identity of that order determines which routine will be entered next.

If the order encountered is the third type of delete it will only cause deletion of a descriptor whose batch number matches the batch number contained in the order itself. The WCT orders are read into AB from the WCT; the descriptors are in LM, with the one being checked gated to N and E. From those registers (AB and N), the batch numbers are gated to SAL for comparison. An equal compare causes the descriptor to be deleted as before; its symbol count is added to K, and the ODT address is stepped plus two. As in the previous delete routine, the next WCT order determines the routine to be entered next.

Prior to entering the next routine, however, the microprogram converts the accumulated symbol count from a word count to a byte count and adds it to the old refresh memory (ORM) address in GPR 4. This deletes the symbols from old refresh memory, by stepping the ORM starting address ahead to a point past the last deleted symbol. When the symbols are moved from ORM to NRM, the first symbol moved is fetched from the address in GPR 4.

If no interrupt is pending, the move history descriptors routine will be entered next. A STAT trigger is turned on and indicates that the microprogram is moving history descriptors. (This routine is also used to move current descriptors.) The first descriptor to be moved from the ODT is in N and E from the previous routine. If it is not a null, the descriptor is gated to ST for storing at the first NDT address (obtained from GPR 3). Movement to ST is accomplished in two steps for each descriptor. The path is from N, through the serial adder, and into ST per the STC. Two steps are required to move one descriptor because the adder is only one byte wide. N(0–7) is gated first; then, after STC is stepped plus one, N(8–15) is gated into ST. After the descriptor has been gated to ST, the symbol count E(8–15) is added to K to accumulate the history symbol count. If the descriptor was gated into the last halfword position of ST (bits 48–63), the doubleword in ST is stored in the NDT at this time. If ST is not full, however, the next descriptor from the ODT is gated from LM into N and E, and the process of gating the descriptor from N to ST is repeated. After each descriptor is gated from LM to N and E, LM is checked to see if the descriptor was gated from the last halfword position (bits 48–63). If it was, LM is empty, and a new doubleword containing four descriptors is read from the ODT into LM. This process of moving descriptors from LM to ST, refilling LM from the ODT when all the descriptors have been gated from it, accumulating the history count, and storing ST at the current NDT position when it is full of descriptors continues until a null descriptor from the ODT is encountered.

With no interrupt pending, the next WCT order is checked; if it is not a Modify order, the null descriptor is stored following the last history descriptor, and the modify current descriptors and move current descriptors routines are bypassed. The null indicates the end of history descriptors in the NDT, and the accumulated history count is stored as a current count in GPR 1. Note that, if there was no Modify order, the "history" descriptors that have been moved are really current descriptors since single-symbol data is all "current".

If the next WCT order is a Modify order, the current descriptor following the null in the ODT is examined. If its batch number is the same as that of the Modify order, the descriptor (now history) is stored in the NDT, and its symbol count is added to the accumulated history count in K. The descriptor gating is the same as it was in the move history routine, from N through the serial adder (in two steps), into ST, and to the NDT. After the descriptor has been stored in the NDT, the next WCT order is checked. There can be more than one Modify order in the WCT; as long as the current order is a Modify, the last routine is repeated.

When the microprogram finds that the WCT order is not a Modify, it branches to a routine which saves the accumulated history (and modified current) count in GPR 0 and generates and stores a null descriptor following the last descriptor stored in the NDT. The NDT now contains all of the history descriptors and the null separator. RPSB then branches to the same routine that moved the history descriptors, to begin moving all the current descriptors that remain in the ODT after the Modify orders have been completed. The routine is executed exactly as before, but the STAT trigger being off, this time, is the indication that current descriptors are being processed. After all current descriptors have been moved (up to the class type null descriptor), the STAT trigger off causes a branch to a test that checks the next WCT position.

The next order can be an end of class type (EOCT) order (there may be no new input data to insert on some executions of RPSB). If it is an EOCT, a null descriptor is stored following the last current descriptor in the NDT. This acts as an end of class type separator and completes the assembly of the NDT for this class type of input data. All the descriptors have been moved, and the section of the instruction that moves the display words (referred to as symbols) from ORM to NRM is entered.

If the WCT order is not an EOCT, it must be an Insert Descriptor order. If it is not, condition code 3 is set to indicate an illegal sequence in the WCT, and the execution is terminated.

An Insert order indicates that new input data has been processed by CSS, and it must be added to new refresh memory (NRM) as additional current data. The Insert Descriptor order causes RPSB to construct a descriptor for the new data and store it in NDT following the last current descriptor. The batch number assigned to the new descriptor is taken from bits 8–15 of the insert Descriptor WCT order. The other half of the descriptor, the symbol count,

must be calculated by using two sort bin addresses furnished by software. The sort bin start displacement in GPR 9 addresses the first new display word in the sort bin. The displacement from the bin displacement table associated with the given Insert order in the WCT is the stop displacement of the sort bin data (i.e., the displacement of the last display word in the sort bin for the given batch). Essentially, the insert routine subtracts the start displacement (GPR 9) from the stop displacement to obtain the symbol count. The batch number is gated to ST from N(8–15) to become the first half of the new descriptor. The count is gated into ST to form the second half of the new descriptor and is also added to K to begin the accumulated insert count.

After storing the new descriptor in the NDT, following the last current descriptor, the next WCT order is read into LM. After each insert order, the WCT address is updated by adding 36 to it to locate the next order. This must be done because of the way the WCT is formatted. A bin displacement table follows each insert order (which must always be aligned on an odd-word boundary).

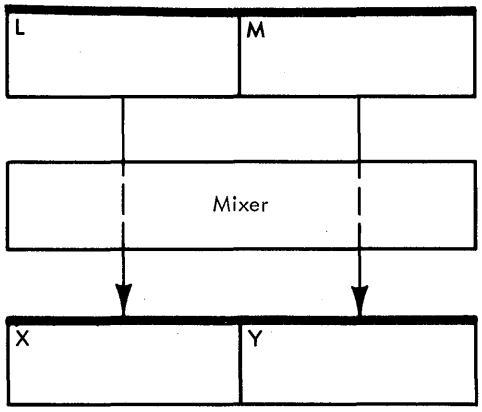
If the next WCT order is an EOCT order, the routine mentioned previously is entered to store a null descriptor after the last descriptor in the NDT and to prepare to move symbols from ORM to NRM. The next order can be another Insert Descriptor order, however. In this case, the microprogram repeats the insert routine just discussed. The next order could also be a Modify. A Modify order here would provide for insertion of a history/current null separator. This is needed in cases where the initial descriptor table is being built and there is no ODT. The previous Insert orders in this case were used to insert history descriptors. As soon as an EOCT order is encountered at this point in the microprogram, a complete class type has been processed. This means that the NDT is completed for that class type, and the microprogram branches to the routine that begins initialization for the moving of symbols from ORM to NRM.

Symbols Section

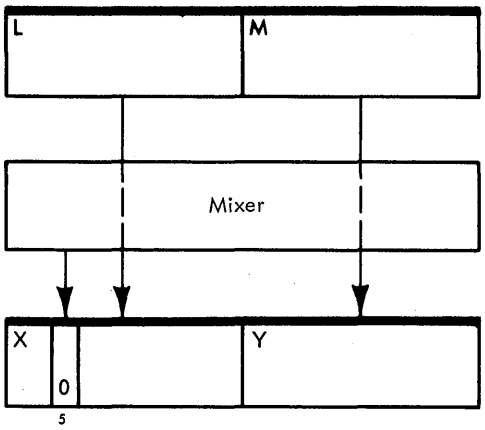
This section begins by moving those symbols still to be displayed from ORM to NRM. The ORM address, updated in the last section, is the address of the first symbol to be moved; the NRM address (GPR 5) is the location at which it will be stored. The microprogram uses IC to hold the source address (ORM) and D to hold the destination address (NRM). To keep track of which symbols are being moved, the history, current, and insert counts accumulated in the previous section are decremented, in turn, as the corresponding symbols are moved.

The data from ORM is read into LM (per IC), gated through the mixer into XY, and stored from there into NRM (per D). The type of data being gated through the mixer determines the effect the mixer has on the data. Two micro-orders, FMTO*E13–15 and FMTN*E13–15, are issued at the proper times to gate data thru the mixer. FMTO*E13–15 is used when data from ORM is being stored into NRM. When the new input data from a sort bin is being gated from LM, the FMTN*E13–15 is issued. E13–15 is set to the value that will cause the desired result when the micro-order is issued. A list of the ten possible FMTO and FMTN micro-orders and their purposes is presented here.

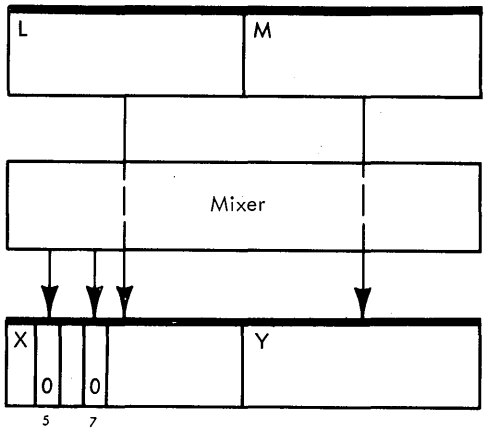
FMTO–0: Gates LM to XY with no changes. Used when gating current data from ORM to NRM.



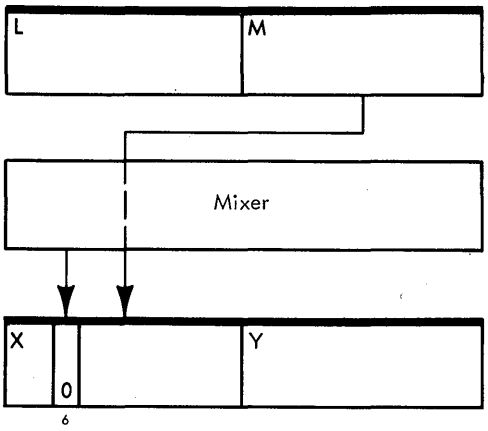
FMTO–1: Gates LM to XY and turns off the BR 1 bit (bit 5). Used when the doubleword of data in LM contains the last history or modified current symbol in the left word and the first current symbol in the right word.



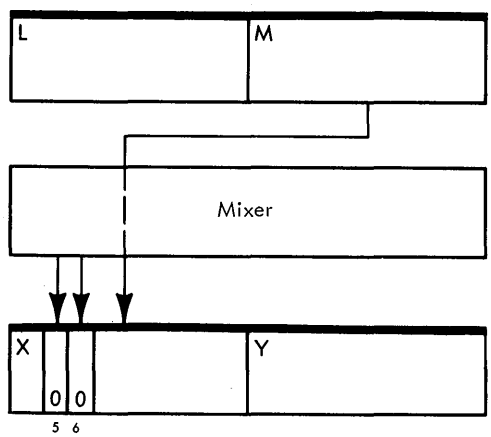
FMTO–2: Gates LM to XY and turns off both BR bits (bits 5 and 7). Used when gating history data from ORM to NRM or when modifying current data.



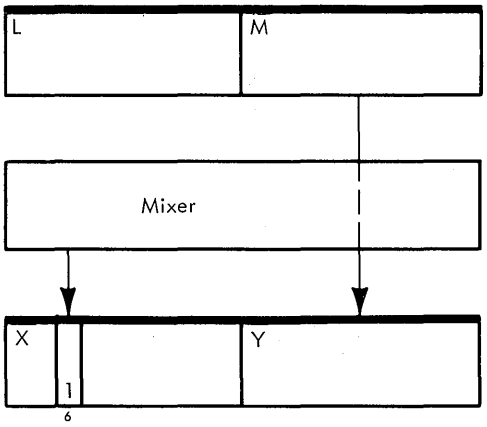
FMTO–4: Gates M to X and resets the P2 bit (P2 bit off indicates that target 2 is not to be displayed). Used for the first current symbol if on an odd-word boundary and there is no history.



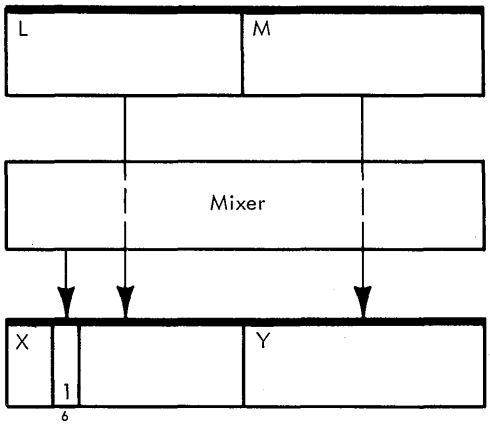
FMTO–5, 6: Gates M to X, resets the P2 bit, and resets BR 1 bit (bit 5) in X. Used for the first history symbol if on an odd-word boundary (following an odd delete operation).



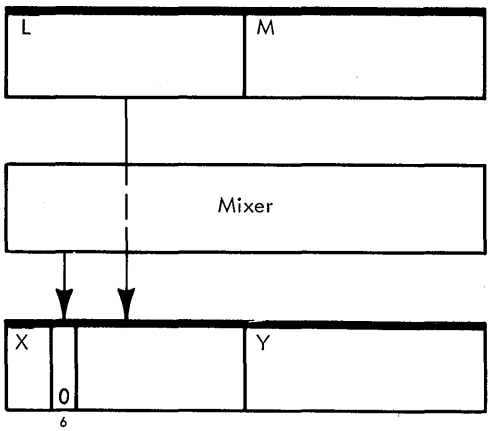
FMTN–0: Gates sort bin data word from M to Y and turns on P2 bit to cause the righthand symbol to be displayed. This format order should follow FMTN–2 or FMTN–7. It may be used alone initially if the first sort bin word is taken from an odd word boundary and has to be packed.



FMTN–1: Gates both sort bin datawords from LM to XY and turns on the P2 bit. It is used when both the sort bin address and the NRM address are on even word boundaries.

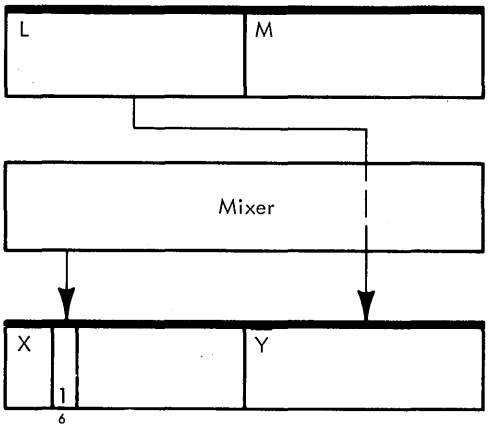


FMTN–2: Gates L to X and resets P2 bit. Used to store the last insert symbol when both the sort bin address and the NRM address are on even word boundaries.

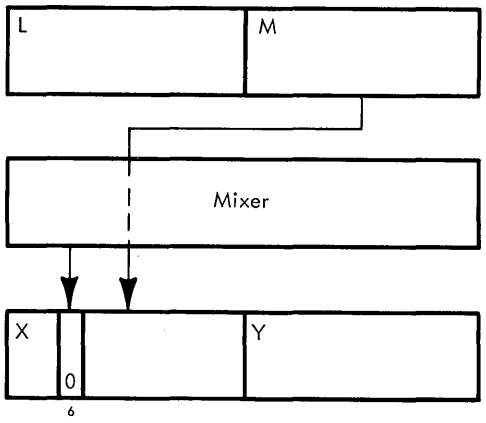


FMTN–5: Forces the P2 bit (bit 6) to 0 and corrects bad parity (erroneously generated when an FMTN–0 order follows either an FMTN–7 or an FMTN–2 order).

FMTN–6: Gates L to Y and turns on P2 bit. Used when the sort bin data address is on an even word boundary and the NRM address is on an odd word boundary. This format order should follow FMTN–7.



FMTN–7: Gates M to X and resets the P2 bit. Used when sort bin data address is on an odd word boundary and NRM address is on an even word boundary.



The first routine reads ORM data into LM (per IC) and gates the history count from GPR 0 to T and B. If the history count is greater than two, E13–15 is set to 2 and FMTO*E13–15 gates the doubleword to XY. After the doubleword is stored in NRM, the count is decremented, and both ORM and NRM addresses are updated. This routine is repeated until the history count decrements to 1 or 2. When that occurs, it is determined where the last history word is: the word in L (count=1), or the word in M (count=2). In each case, the first current symbol will be in a different location, and the microprogram must make sure these symbols directly follow the history symbols in NRM.

The count and the boundary of the ORM address (doubleword or word) indicates which FMTO micro-order must be issued to gate the data properly to XY. After this is done, STAT D is turned on to indicate that current data is now being moved from ORM to NRM. This same routine is used to process the current data. The operation is very similar to moving history data, with the main difference being the values set in E13–15. Different FMTO*E13–15 micro-orders are used because, in this case, the BR (brightness) bits must not be reset. The current count was gated into B, and, when it decrements to 1 or 2, a routine similar to the one used at the end of history symbols is entered to store the last current word in the correct position in NRM. The symbols to be inserted from the sort bin must be contiguous to the current symbols because they are additional current symbols.

All of the data to be retained has now been moved from ORM. The remaining portion of the microprogram moves the symbols from the sort bin into NRM. The FMTN*E13–15 micro-orders are used to gate the sort bin data, read into LM, through the mixer into XY for storage in NRM. If the sort bin address and the NRM address are on the same boundaries, the transfer of symbols is very easy. The new symbols are gated straight through the mixer (L to X and M to Y) by the FMTN*E13–15 micro-order. However, when the sort bin data is on an odd boundary and the NRM address is on an even boundary, the data in LM must be aligned to an even-word boundary before it can be stored in NRM. This is accomplished by gating the first sort bin word from M to X, reading the next doubleword (per IC) from the sort bin into LM, and gating the second sort bin word from L to Y. The data is now correctly aligned in XY and is stored in NRM. The third sort bin word, now in M, is gated through the mixer into X (as was the first one), and another doubleword is read into LM from the sort bin. This process continues until the insert count reaches 0. If the last sort bin symbol is in X, a blank symbol is gated into Y, and the P2 bit is reset. Thus, NRM will always end on an even boundary.

NRM is now completely built for one class type, and the microprogram branches back to check the next WCT order. An EOB order encountered here indicates that there are no more class types to process, and the instruction is terminated. If the WCT order is not an EOB, the instruction recycles to process another class type, and the microprogram branches to the routine that checks for Delete Descriptors orders.

To honor a pending interrupt during execution of RPSB, a branch is taken to a routine which sets codes (INTRP ID) to allow the instruction to be re-entered at the point of interruption.

If RPSB was operating in the descriptors section of the microprogram when the interruption occurred, INTRP ID bits 4–7 are set to a value which, when interrogated upon re-entry, cause the re-entry to be at the beginning of the routine about to be entered before the interruption occurred.

The following are the settings of bits 4–7 when interruption is allowed in the descriptors section.

1. Set to 1, at the end of the delete descriptors routines. Re-entry will be at the beginning of the move history descriptors routine.
2. Set to 2, at the end of the move history descriptors routine. Re-entry will be at the beginning of the modify descriptors routine.
3. Set to 4, at the end of the modify descriptors routine. Re-entry will be at the beginning of the move current descriptors routine.
4. Set to 6, at the end of the move current descriptors routine. Re-entry will be at the beginning of the insert new descriptors routine.

Note: When an interrupt is allowed at the end of the insert new descriptors routine, INTRPT ID bits 0–3 are set to 9. Re-entry will be at the beginning of the move symbols section of the microprogram.

When RPSB is operating in the symbols section of the microprogram, re-entry is always to the same point, i.e., the routine that initializes for the move from ORM to NRM. For nonprogram interrupts, the symbol counts and data addresses are corrected before the interrupt is allowed. The total number of symbols that have been moved from the sort bin to NRM is calculated by subtracting the starting NRM address (saved in FPR 3) from the current NRM address (D). The value of the counts (history in GPR 0, current in GPR 1, and insert in FPR 2) is the original number of those symbols to be moved to NRM; they are used to determine the point of interruption. First, the history count (T) is subtracted from the number of symbols that have been moved (B), and the result is saved in A and D. From this result it is determined if still more history symbols are to be moved. The result of the subtraction will be the number left to be moved. This number is stored in GPR 0 in place of the original history count, and the starting ORM address (GPR 4) is updated by the number of symbols moved. RPSB can be terminated now because re-entry to the ORM-to-NRM initialization routine will now begin moving the proper symbols.

When the number of symbols moved is larger than the history count, all history symbols have been moved and GPR 0 is set to 0. The current count (GPR 1) is then subtracted from the result obtained when the history count was subtracted from the number of symbols moved, and the result is again gated to A and D. This result indicates whether all current symbols have been moved or not. The same procedure used for the history count is used here, and the current count (GPR 1) is set to 0 or to the count that remains to be moved. If all current symbols have been moved, the result of the last subtraction (the number of insert symbols that have been moved) is subtracted from the total number of symbols moved. The result is added to the starting ORM address to update it to the last current symbol moved from ORM. Inserted symbols do not come from ORM. To update the insert count, the insert count moved is subtracted from the original insert count, and the result is stored in FPR 2. The sort bin address (GPR 6) is already pointing to the correct location since it is kept current as the symbols are inserted into NRM.

After the counts have been corrected, bits 0–3 of the INTRP ID (GPR 8) are set to 9. This indicates, when RPSB is reissued, that entry is to be to the routine that begins initializing for the move of symbols from ORM to NRM.

RPSB checks for two specific types of interrupts: nonprogram interrupt (I/O or external), and program (addressing storage protection, and specification). For any nonprogram interrupt, the data counts and addresses are corrected, the instruction address (360 IC) is adjusted by -2 to point back at this RPSB instruction, and Q, R, and E refilled before instruction termination. Thus, interrupt recovery is automatic for nonprogram interrupts.

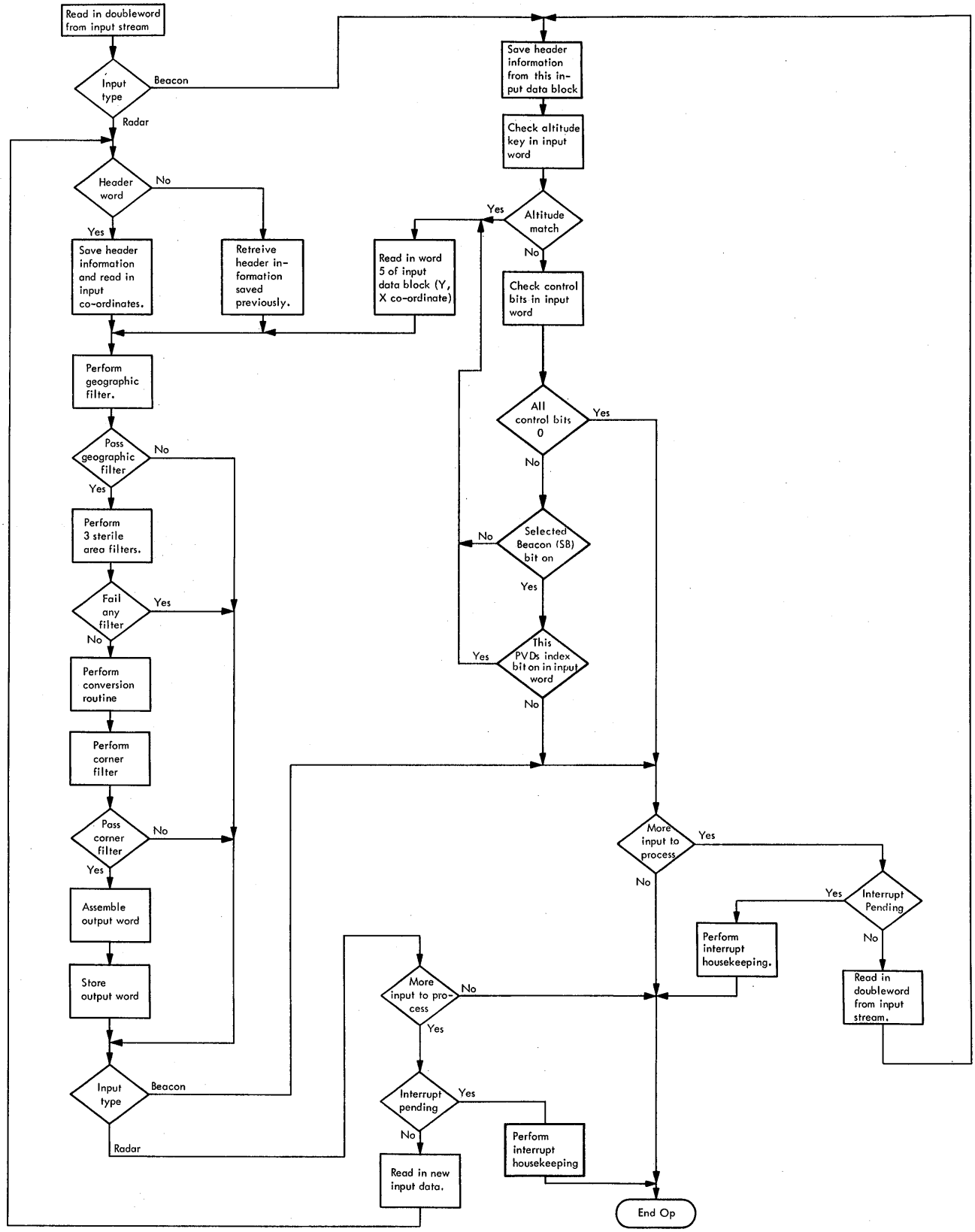
For program interrupts, however, interrupt recovery, if possible at all, is not automatic. The instruction address (360 IC) is not adjusted to point back at the instruction, nor are the data counts and/or addresses always corrected. For those program interrupts that are recoverable (certain invalid DE addressing and storage protection exceptions), the interrupt routine must provide the necessary adjustments for recovery.

Time clock update interrupts are detected at the same interrupt points as other interrupts, but they normally do not cause instruction termination and instruction interruption (swapping of PSWs). A time clock update request is handled by an internal RPSB routine which updates the clock; then, if there are no pending interrupts (including an external interrupt generated when the clock is stepped from + to -), RPSB returns control to the proper routine via the INTRP ID code.

CONVERT AND SORT SYMBOLS, CSS (02)

- Submits an input stream of primary radar or beacon data to one PVD's geographic and sterile area filters.
- Converts the input coordinates that pass all filters from system scale to PVD scale.
- Sorts input coordinates into 16 sort bins, according to each coordinate's location on the PVD face.
- Assembles and stores an output word for each input target that passes all filters.
- Control information is furnished by software and is contained in GPRs and FPRs.

- RR Format


Read in doubleword from input stream
Input type
Beacon
Radar
Header word
No
Yes
Save header information and read in input co-ordinates.
Retreive header information saved previously.
Perform geographic filter.
Pass geographic filter
No
Yes
Perform 3 sterile area filters.
Fail any filter
Yes
No
Perform conversion routine
Perform corner filter
Pass corner filter
No
Yes
Assemble output word
Store output word
Input type
Beacon
Radar
Save header information from this input data block
Check altitude key in input word
Altitude match
Yes
No
Read in word 5 of input data block (Y, X co-ordinate)
Check control bits in input word
All control bits 0
Yes
No
Selected Beacon (SB) bit on
No
Yes
This PVDs index bit on in input word
Yes
No
More input to process
Yes
No
Interrupt Pending
Yes
No
Perform interrupt housekeeping.
Read in doubleword from input stream.
More input to process
No
Yes
Interrupt pending
Yes
No
Perform interrupt housekeeping
Read in new input data.
End Op

- Program Interrupts
  1. Protection (store or fetch protect violation).
  2. Addressing (input or output addresses outside available storage).
  3. Specification (new page address, contained in bytes 5–7 of doubleword following a 512-byte page, is not on a doubleword boundary).

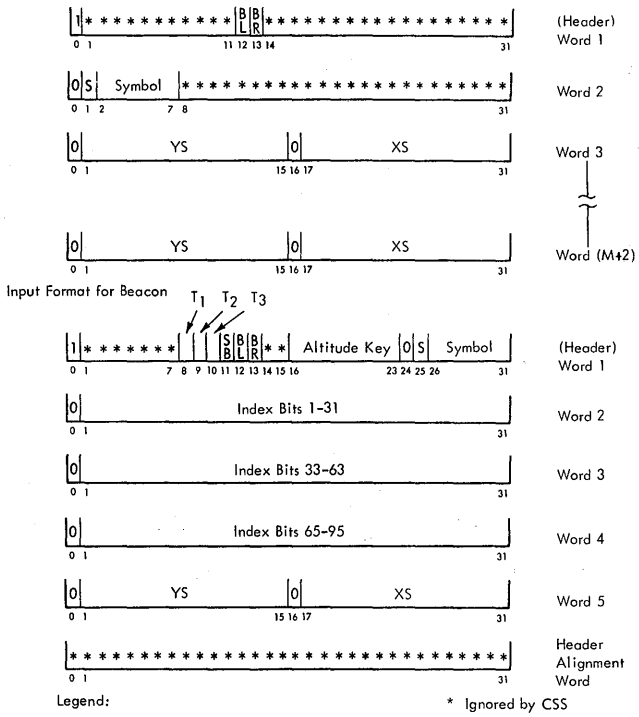
- Condition Code
  0 Processing complete, no data stored in any sort bin.
  1 Processing complete, data stored in one or more sort bins.
  2 Sort bin page overflow. A new page is needed for the bin whose number is in GPR 9 bits (4–7).
  3 Invalid beacon format; the first word of a data block was not a header.

- Detailed discussion is divided into two sections: (1) "Primary Radar/Single Symbol Input" and (2) "Beacon Input".

- Refer to Diagram 5-903, FEMDM.

The CSS instruction processes either a primary radar/single symbol input data stream or an input data stream of beacon data blocks.

Input Format for Primary Radar/Single Symbols.



Input Format for Beacon

Legend:

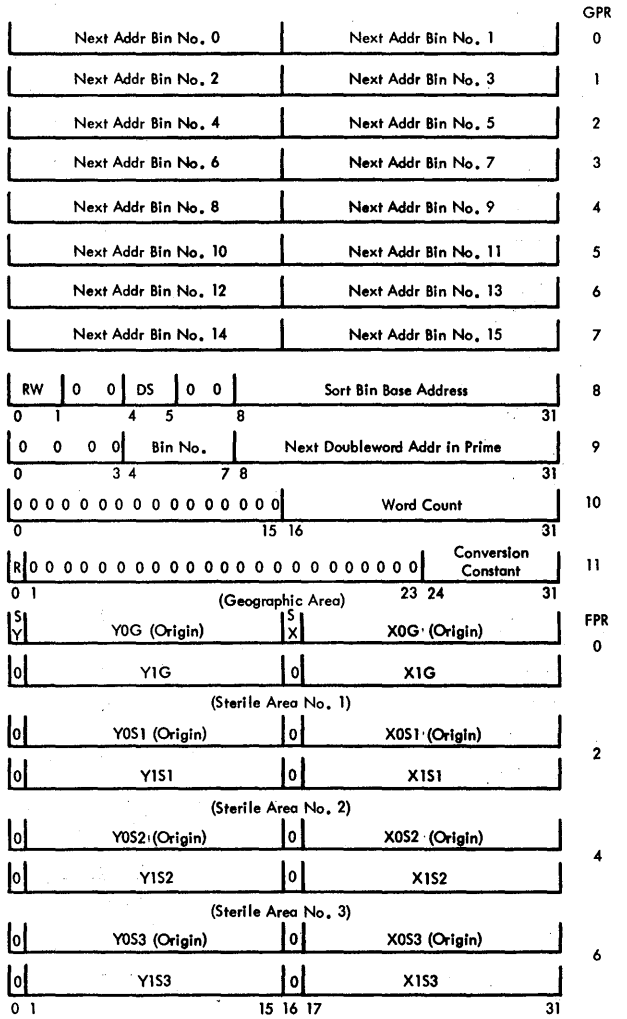
BL Blink
BR Brightness
S Symbol Size
T_1, T_2, T_3 Type bits
SB Selected Beacon
M Number of X, Y position words in primary radar/single-symbol data block (if necessary, a dummy position word of all zeros must be included to make the count even)

* Ignored by CSS

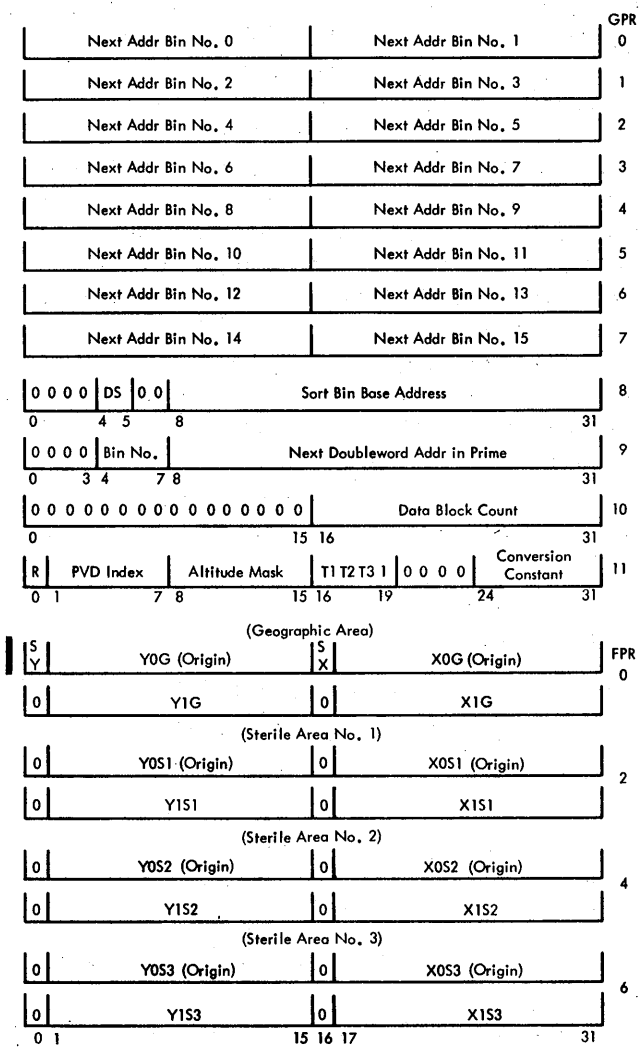
Note: Multiple data blocks accepted for input.

The instruction is used to prepare display data for use by the Repack Symbols instruction and must be issued once for each PVD. It selects the input coordinates that pertain to the PVD being serviced from the input data stream by means of geographic, sterile area, and corner filters. Beacon data can be subjected to additional filtering (altitude, type, and index). The coordinates that pass all these filters are converted to the scale of the PVD and stored, according to their positions on the face of the PVD, in one of 16 storage areas designated as sort bins. The control information required by CSS to perform these tasks is furnished by software (in the form of data stored in the GPRs and FPRs). Specific register contents for the two different input types are listed below, followed by a description of the use of each.

1. Primary Radar/Single Symbol Input:



2. Beacon Input:



Next Addr Bin #N–(GPRs 0–7): These are halfword operands, each of which is a displacement value (which must be greater than hex 27, less than, or equal to hex FE00, and may not have the values hex 1F8 or 1FC in the low-order nine bits). When the operand is added to the sort bin base address (GPR 8), the sum defines the next word address in the n'th sort bin, into which the output data word may be stored. After each output data word is stored, the halfword operand displacement is incremented by four bytes.

RW–(GPR 8–Primary Radar/Single Symbol Data Only): The RW bits 0 and 1 (a right-word indicator) are normally set to 0 by the programmer. Whenever (in processing primary radar/single symbol data) a sort bin page overflow condition occurs after processing of the left word (X, Y position) of a data doubleword, the RW bits are set. The instruction then terminates. Upon reissuance of the given instruction, the RW bits cause CSS to refetch the original data doubleword (containing the left-hand word already processed) and to begin by processing the right-hand data word.

DS–(GPR 8): The data stored (DS) bits 4 and 5 are reserved for use by CSS and are normally set to 0 by the programmer. Prior to any termination due to I/O or external interrupt or bin page overflow (word count not 0), the DS bits are set if any output words were stored in any sort bin in CSS execution up to this point. They indicate to CSS (upon re-entry from interrupt or bin page overflow handling) that output data was stored in a previous CSS execution and that CC 1 must be set to 1 upon normal termination of CSS (word count 0), whether or not such termination occurs in the current CSS execution or subsequent CSS executions (after a series of interrupts or bin page overflow conditions). These bits are reset by normal termination (word count 0) or by any program interrupt.

Sort Bin Base Addr–(GPR 8): This address must always lie on a N512 + 8 page boundary (i.e., the low-order nine bits must contain the hex value 8). It is used as the base address for referencing the 16 sort bins.

Bin No.–(GPR 9): When CSS encounters a sort bin page overflow (126 output words have been stored in a sort bin page), it stores the affected sort bin number in bits 4–7 of GPR 9 (Bin No.). CSS then terminates with CC 2.

Next DW Addr In Prime–(GPR 9): Bits 8–31 maintain the address of the next input doubleword to be accessed from storage and are updated on a doubleword basis as each doubleword is fetched.

Input Data Count–(GPR 10): GPR 10 initially must contain the input data count for the data to be processed by CSS. This count is subsequently maintained in GPR 10 by CSS during the instruction processing. The significance of the input data count depends on the type of data to be processed, as shown below:

1. Primary radar/single symbol: The input data count must be even and must reflect the number of words to be processed, i.e., the number of data words (Y, X coordinates) plus two for each header doubleword. This count is decremented by one for each data word processed and by two for each header processed.
2. Beacon: The input data count must reflect the number of beacon data blocks to be processed. CSS will decrement this count by one upon completion of processing for a data block (i.e., each six words).

R–(GPR 11): Bit 0 in GPR 11 is a control bit affecting the brightness bit (BR) in every output word generated by CSS. If R is 0, the BR bit in every output word is taken from bit 13 (BR bit) of the header last encountered in the input data stream; if R is 1, the output BR bit is made 0.

PVD Index–(GPR 11–Beacon Data Only): The PVD Index field is contained in bits 1–7. It addresses the index bits in words 2–4 of the beacon data block.

The PVD INDEX value (bit displacement from bit 0, word 2) is as follows:

PVD Index Bits	Code		
1, 2	00	Word 2	
	01	Word 3	
	10	Word 4	
	11	Invalid	
3, 4	00	Byte 0	of selected word
	01	Byte 1	of selected word
	10	Byte 2	of selected word
	11	Byte 3	of selected word
5–7	000	Bit 0	of selected byte
	001	Bit 1	of selected byte
	010	Bit 2	of selected byte
	011	Bit 3	of selected byte
	100	Bit 4	of selected byte
	101	Bit 5	of selected byte
	110	Bit 6	of selected byte
	111	Bit 7	of selected byte

Altitude Mask–(GPR 11–Beacon Data Only): Bits 8–15 specify altitude filtering ranges. Each bit corresponds to a different altitude range.

Type Filtering Bits–(GPR 11–Beacon Data Only): Bits 16, 17, 18 of GPR 11 are a type mask (T1, T2, T3).

Conversion Constant–(GPR 11): The conversion constant (bits 24–31) comprise a scaling control constant (bits 24–25) and a conversion multiplier (bits 26–31). The conversion constant is used to convert coordinates, relative to the PVD origin, from system scale to PVD scale. (The translation of system coordinates from system origin to PVD origin is performed during geographic filtering.)

Conversion constants may be derived from the following formulae:

$$M = \frac{4096}{S' \times 4^N} \quad (a)$$

$$S = \frac{4096}{M \times 4^N} \quad (b)$$

where S' is the nominal scale (PVD diameter in nautical miles).

S is the actual exact scale truncated to the nearest sixteenth of a mile.

M is the conversion multiplier and must be an integer in the range $1 \leq M \leq 42$.

N is the scaling control constant and must be an integer in the range $0 \leq N \leq 2$.

Given any nominal scale S', formula (a) may first be used to solve for any acceptable combination of M and N with M rounded to the nearest integer. For example in solving for S'=28, N=0 would indicate M=146, which is invalid; for S'=28, N=1 would indicate M=37, which is acceptable; for S'=28, N=2 would indicate M=9, which is also acceptable but a poorer approximation.

The actual exact scale S should then be determined, using the M and N values from formula (b). (Here, S must be truncated to the nearest sixteenth of a nautical mile, less than the actual quotient.) Continuing our example, in solving for M=9, N=2 (or M=36, N=1) would indicate S=28.44 (28 7/16<28.444); for M=37, N=1 would indicate S=27.63 (27 10/16<27.67). Note that in solving for M=16, N=0 would indicate S=255.94 (255 15/16 <256.00).

The following table lists the standard set of conversion constants used in the 9020E system. These are truncated to the nearest sixteenth of a mile.

SCALE		CONVERSION CONSTANT	
Nominal	Exact	Binary	Hex
12	12.19	10010101	95
18	18.25	10001110	8E
28	27.63	01100101	65
43	42.63	01011000	58
64	63.94	01010000	50
102	102.38	00101000	28
128	127.94	00100000	20
158	157.50	00011010	1A
205	204.75	00010100	14
256	255.94	00010000	10
315	315.06	00001101	0D
410	409.56	00001010	0A
512	511.94	00001000	08
819	819.19	00000101	05

GPRs 12–15: These registers may be used by the programmer.

FPRs 0, 2, 4, 6: The doubleword floating-point registers define the PVD geographic area and the three sterile areas (areas in which radar and single symbol data cannot be displayed) in terms of 15-bit system coordinates given relative to the system origin. Each area is a rectangular area (square for PVD) defined by two sets of X, Y coordinates; one represents the lower-left corner, the other the upper-right corner.

Each coordinate is specified to the nearest sixteenth of a nautical mile over the range 0-2047.9375; i.e., the binary decimal point is always assumed to be between bits 11 and 12 or bits 27 and 28 (for the Y and X coordinates, respectively).

When the input to the instruction is beacon data, additional checks are made to see whether the input coordinates are for the PVD being processed. These checks are performed on bits 8–23 of the first input word (header).

PVDs often must display only the targets that are within a given altitude range. The altitude key in the header word (bits 16–23) gives the target's altitude; this is compared with the PVD's altitude mask (in GPR 11). An altitude match causes CSS to enter the geographic filter, directly bypassing both type and index filtering; if all remaining filters are passed (geographic, sterile area, and corner), this data is to be displayed. If there is no altitude match, the type filter is entered.

Type-filtering is a comparison between type bits 8–10 (T1, T2, T3) of the header word and the mask provided in GPR 11, bits 16–18. A match on any bit position causes CSS to enter the geographic, sterile area, and corner filters, bypassing index filtering. A no-match condition causes CSS to enter the index filter and to check the selected beacon bit (SB) in the header word. The selected beacon bit being on indicates that this input data block contains beacon data for selected PVDs. Every beacon data block contains three index words, which are used to indicate the selected PVDs. Each index bit in input words 2, 3, and 4 is assigned to one particular PVD. A particular PVD is addressed by means of the PVD index located in GPR 11. If the addressed index bit is on, the selected beacon data is valid for a test by the remaining filters.

CSS execution begins by reading the first input doubleword from the input stream. The address of the first input doubleword is read from GPR 9 per the R2 field. (Since the input address is located in GPR 9, the R2 field in the instruction format must always contain the value 9.) After the input doubleword is gated into AB, the R1 field is checked. The value of the R1 field indicates the type of input contained in the input stream: 0 denotes primary radar/single symbol input, 2 denotes beacon input.

Primary Radar/Single Symbol Input

As soon as it is determined that radar/single symbol input is being processed, the header information at the sort bin base address plus 8 is fetched. This is done in case this is a re-entry to execution after an interrupt has been honored and the first doubleword read from the input stream is not a header doubleword. (When an interrupt occurs before CSS execution is complete, the instruction saves the current header information in a CSS work area (sort bin base address plus 8) before the interrupt is allowed.) On initial entry (which includes interrupt or bin paging re-entry), the constant 9 is set into F(4–7) to indicate to the microprogram, in the next routine, that entry to that routine is a result of the instruction being issued (rather than an internal recycling back into it after the processing of an input doubleword).

If the right word indicator (RW) is off after gating in a new input doubleword, the left word must be processed. Bit 0 of A is gated to SAL in preparation for a four-way branch to determine whether the input is a header or a data word. On initial entry (when F4–7 = 9), the header information fetched from storage is gated from SDBO to the LSWR (via T), where it is kept until it is needed for assembly of an output word. If the doubleword from the input stream in AB is a header, the new header replaces the old one in LSWR. The conditions that determine the routine to be entered next are the type of entry (initial or recycling) and whether the input word is a header or a data word. If the input is a header, the symbol, S, BL, and BR bits are assembled into one word and stored in LSWR to replace the old header information. After updating the input count, if there are no pending interrupts, the next doubleword is read from the input stream. The microprogram then branches back to the routine which examines the next input doubleword.

If the right word (RW) indicator is on after a new input doubleword is gated in, the right word must be processed. The header information fetched from memory is gated from SDBO to the LSWR. It is known that this is a re-entry since the RW indicator normally should never be on upon initial entry to the instruction. RW on also indicates that the input is data.

When the input is a dataword, whether upon initial entry or re-entry, the brightness control bit (R) (in GPR 11) is checked. This bit indicates to the microprogram, if it is a 1, that the BR bits in all output words assembled for this PVD must be 0's. STAT H is now set if the brightness control bit is off, and it becomes a branching condition to the microprogram during assembly of the output word. The geographic and sterile area filters routine is entered next, with the coordinates to be checked in the A register.

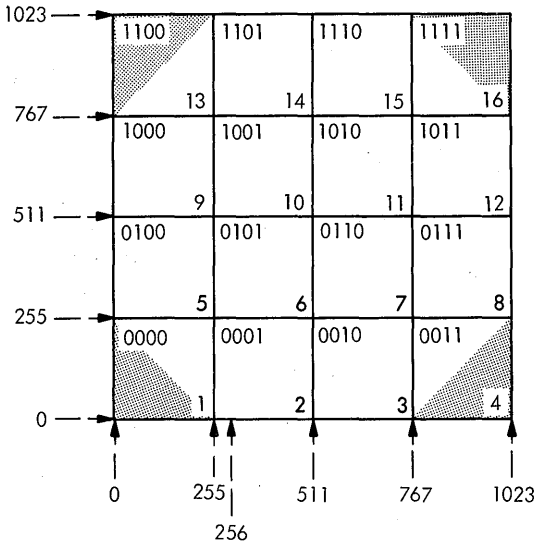
This routine consists of four filters: the geographic filter, the sterile area 1 filter, the sterile area 2 filter, and the sterile area 3 filter. Failure to pass any of the four filters causes the data word being processed to be rejected. If the left word is rejected, the right input word is gated into A, the input count is updated, and the microprogram branches to the beginning of the filter routine. However, when a right word is being processed and is rejected, a new input doubleword is read from the input stream. If no interrupts are pending and the data count has not been decremented to 0, the microprogram reads in the new input doubleword and branches back to the routine which determines whether or not the new data is header information.

If the input coordinates pass all four filters, the microprogram branches directly to the routine which converts the coordinates from system scale to PVD scale.

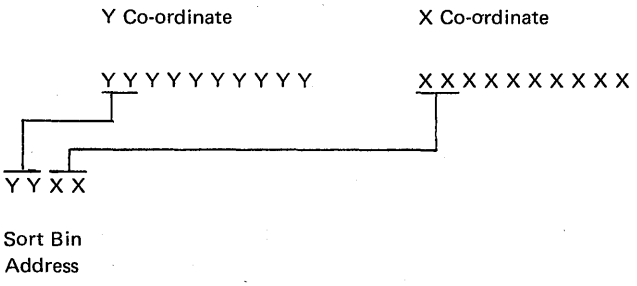
For a discussion of the geographic and three sterile area filters and of the conversion routine, refer to "Introduction to Display Instructions".

The corner filter routine is entered after conversion of the coordinates is completed. The 16 sort bins form a square area which represents the display face of a PVD. The actual PVD display face is round, however, so portions of

the four corner sort bins are not actually displayable. The corner filter rejects input words whose coordinates designate a point in one of these areas.



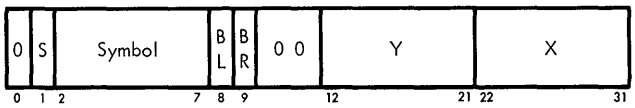
The Y- and X-coordinates are each ten bits long. The two high-order bits of each coordinate are combined to form the address of the sort bin in which the point is located.



These two bits of the X-coordinate are checked first. If they are 00 or 11, it is possible that the point they designate will be in a corner bin. If the high-order X-bits are 11, the point can only be in corner bins 4 or 16. For these two corners, the eight low-order X-bits are complemented (1's complement) so that the routine used to check the other two corners can also be used to check these two. If the high-order X-bits are not equal to 00 or 11, the data word is accepted for display since the designated point cannot be in a corner.

The two high-order Y-bits must be checked if it is determined that the target could be in a corner bin. The two high-order Y-bits are checked in the same manner as the X-bits. If they are 00 or 11, the designated point is definitely in a corner sort bin. The eight low-order bits of the X- (complement of X if target is in sort bin 4 or 16) and Y-coordinates must now be added to find the location within the sort bin. A carry out of the high-order bit of SAL indicates that the sum of the coordinates is 256 or greater. The target is displayable if the carry does not leave all of the SAL bits 0 (e.g., when sum = 256) and the routine that assembles and stores the output word can be entered next. However, if the sum of this addition is less than or equal to 256, the target is located in a corner, and the data word is rejected. If there are no pending interrupts, rejection causes the microprogram to branch back to a prior routine and either start to process the right word or read another doubleword from the input stream.

If the target is not in a corner the output word is assembled in T:



STAT H off causes the BR bit to be reset to 0 as the BL and BR bits are gated to F in preparation for gating to T. The bin address (two high-order Y-bits and two high-order X-bits) was saved in F during the corner filter routine; it is now gated to E(12–15), and it addresses the correct GPR to obtain the sort bin displacement. The sort bin base address is read from GPR 8 after the output word is completely assembled in T. The sort bin base is added to the sort bin displacement, and the result is gated to D to provide the address at which to store the output word (in the next available location in the sort bin). The displacement is updated +4 and stored back in the proper GPR (per E12–15).

Four is added to the address in D after storing the output word, and a check is made for a 512 carry on the addition. A 512 carry indicates a bin page overflow and causes the microprogram to enter the bin page overflow routine (discussed later in this section).

Processing of the input word is now completed. If there is no bin page overflow or pending interrupt, the microprogram fetches the next doubleword from the input data stream.

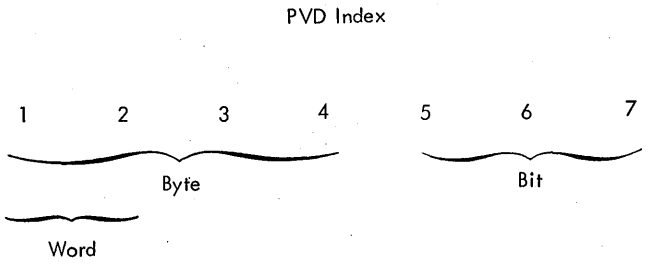
Beacon Input

The first word of each beacon data block must be header information. This word is saved in the LSWR for use in building the output word. Bit 0 of this word (A register) is checked; if it is not a 1, condition code 3 is set, and execution is terminated.

The altitude is checked next by comparing the altitude key (A16–23) with the PVD's altitude mask (GPR 11, 8–15). An altitude match (indicated by SAL = 0) causes the input coordinates to be submitted to the geographic, sterile area, and corner filters. This is accomplished by reading input data block word 5 (Y and X coordinates) into A and branching to the geographic filter routine. This routine is discussed under "Introduction to Display Instructions".

If there is no altitude match, type bits T1, T2, and T3 in bits 8–10 of the header are ANDed with the type mask in bits 16–18 of GPR 11 to determine if there is any type match (i.e., a type bit is left on after the AND operation). If all bits (including the SB bit) are 0, the data block is rejected. If the data block count does not equal 0 and no interrupt is pending, a new header doubleword is read into AB, and the microprogram branches to the beginning routine to process it.

If there is a type match (regardless of the SB bit), the input coordinates are gated into AB, and the microprogram branches to the geographic filter. If there is no type match but the SB bit is 1, the microprogram enters the routine to check the index bit in the input data block (as selected by the PVD index in GPR11). If its index bit is a 1, the PVD has been selected to receive the data. To accomplish this check, the next two words (3 and 4) of the input data block are read into S,T (word 2 is already in B). Bits 1 and 2 of the PVD index (in GPR11 1–7) select the word in which the index bit is located and cause that word to be gated to A (1st or 3rd index word) or to B (2nd index word).



Bits 2, 3, and 4 of the PVD index are set into the ABC counter and gate the byte containing the index bit from A or B to the serial adder bus. The micro-order DECAB decodes bits 5, 6, and 7 of the PVD index (in F5–7) to gate one bit (0–7) from the serial adder bus into the adder. Zeros are gated into the adder for the other six bit positions. SAL is then checked, and, if all-0's, the selected beacon data is not addressed to this PVD. The input data block is rejected, and a new header doubleword is read in to begin processing the next input data block. A non-0 SAL indicates that the index bit is on and that the PVD is selected to receive the input data. The coordinates must still pass all of the filters, however, and the microprogram now reads the input coordinates into A and branches to the geographic filter.

Termination of execution is initiated for one of three reasons:

1. The input count has decremented to 0, indicating the end of the input stream.
2. There is an interrupt pending which must be honored.
3. A bin page overflow has occurred, indicating that software must furnish a new page for the sort bin that overflowed.

When the input stream is completely processed (input count equals 0), a routine is entered to end op. In this routine, the DS (data stored) bits are checked; if they are on, STAT B is turned on, causing condition code 1 to be set. The DS and RW bits are reset in GPR 8 when the sort bin base address (T) is stored back in GPR 8 (8–31). The input stream address was previously updated twice and now must be adjusted -8 (to point at the next doubleword not yet processed) and stored back in GPR 9 (8–31). The contents of GPR 12 and 13, stored in sort bin base address +8 at the beginning of execution, are fetched, and the GPRs are restored. The execution of CSS is terminated by a normal end op after the current header information (LSWR) and any data in K (used by the diagnostic programs) are stored in the CSS work area (sort bin base +8).

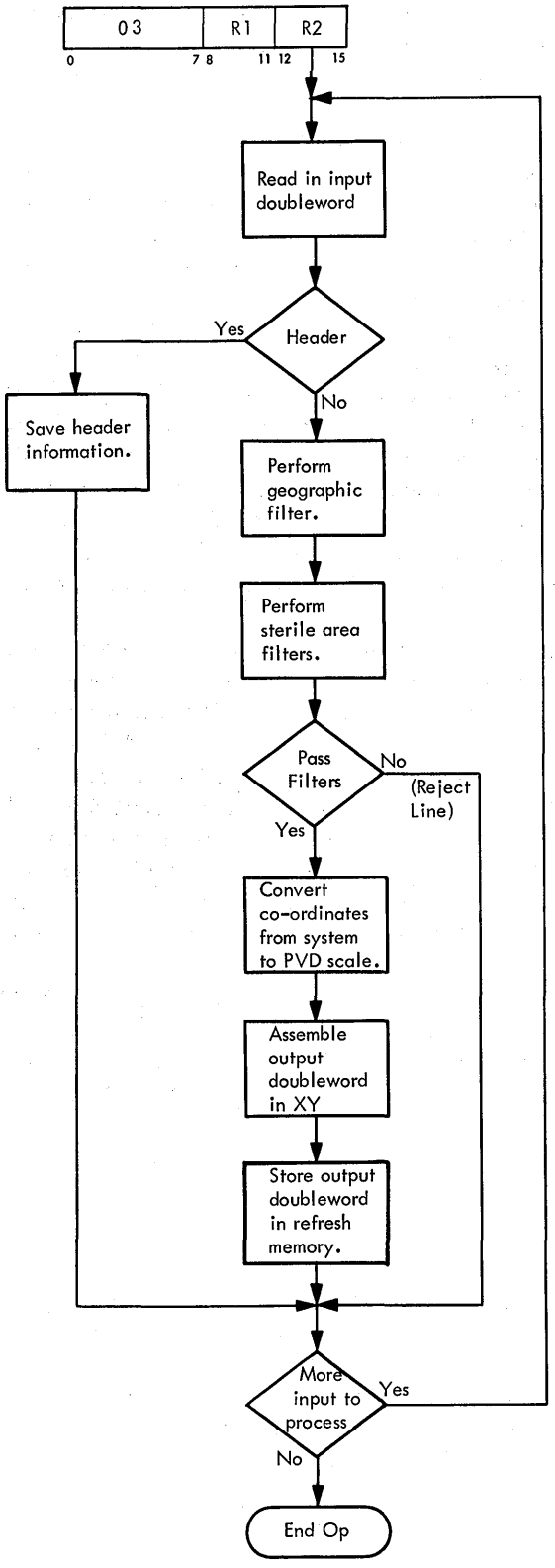
A check for pending interrupts is made after processing is complete on each input doubleword, whether the coordinates are rejected or accepted. When termination is due to a pending interrupt, the data count and the type of interrupt pending affect the action taken. A data count of 0 or a program interrupt cause the microprogram to branch to the normal end op routine. If a nonprogram interrupt (I/O or external) exists, the input stream address is adjusted minus 8, and IC is adjusted minus 2 to point back to the CSS instruction before branching to the normal end op routine.

The bin page overflow routine issues a request for the next doubleword from the input stream (to allow paging), stores the bin number of the bin that overflowed in GPR 9 (4–7), and sets condition code 2 before branching to the normal end op routine.

CONVERT WEATHER LINES, CVWL (03)

- Submits an input stream of weather line coordinates to a particular PVD's geographic and sterile area filters.
- Assembles and formats an output doubleword for each input weather line that passes all the filters and stores it in the PVD's refresh memory.
- Control information contained in GPRs and FPRs.
- Converts input coordinates (system scale) to PVD scale.
- Number of doublewords in input stream is specified in GPR 10.

- RR Format



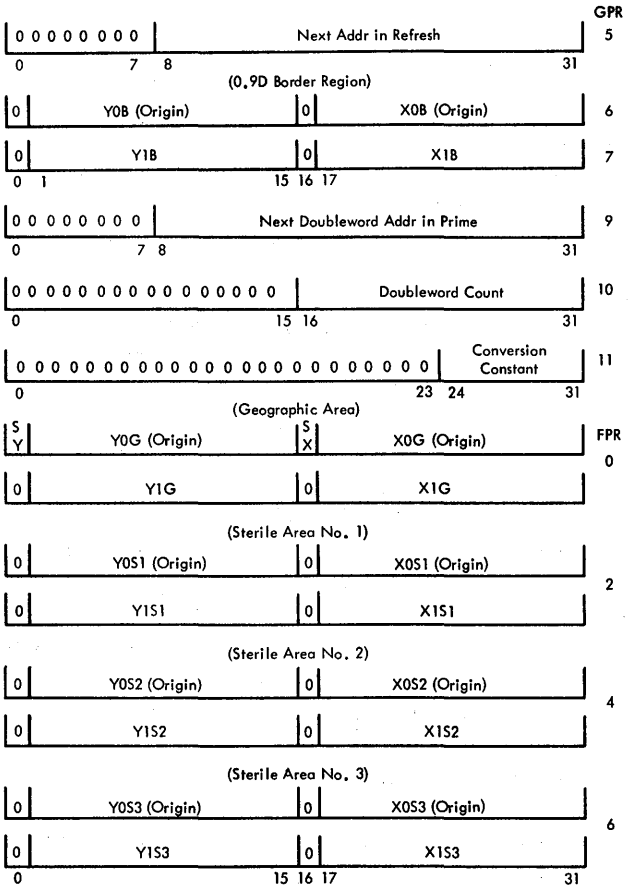
- Program Interruptions
  1. Protection (store or fetch protect violation)
  2. Addressing (input or output addresses outside of available storage)
  3. Specification (new page address, contained in bytes 5–7 of doubleword following a 512-byte page) is not on a doubleword boundary.

- Condition code: unchanged

- Refer to Diagram 5-905, FEMDM.

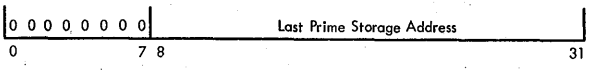
The CVWL instruction processes an input stream of weather line coordinates destined for many different PVDs. The instruction must be issued once for each PVD. It selects (from the input stream) all the weather lines valid for that PVD and stores them in refresh memory as doublewords containing all information required to display the weather lines.

All control information for CVWL is located in GPRs and FPRs. These GPRs must be set up properly (by software) before each CVWL instruction is issued. Control information needed for execution of the CVWL instruction consists of: the geographical boundaries of the PVD, the nine-tenths border-region coordinates of the PVD, the geographical boundaries of each of three sterile areas located within a PVD's geographical boundary, the address from which to read the next input doubleword, the address that indicates where to store the next output doubleword, the input count indicating the number of words in the input stream, and the PVD's conversion constant. Specific register contents are shown below, followed by a description of the use of each.



GPRs 0–2, 4: These general registers are reserved for use as working registers for CVWL.

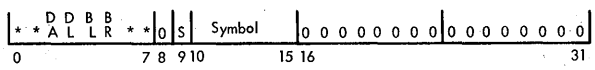
Last Prime Storage Addr–(GPR 3): GPR 3 is used by CVWL to retain the last prime storage data address processed (per GPR 9) if there is a program interrupt. This address is required by the program to restart CVWL after certain program interrupt conditions. The format of this information is as follows:



Next Addr in Refresh–(GPR 5): Bits 8–31 define the next storage location in refresh memory for storing the doubleword weather line result.

0.9 Diameter Border Coordinates–(GPRs 6, 7): X0B, Y0B and X1B, Y1B define the origin (lower-left corner) and the upper-right corner of a usually square area nominally 0.9 times as large as, and centered within, the PVD area defined by positive coordinates in FPR 0.

GPR 8: This general register is used by CVWL to store header information in the event of an interrupt. The format of the header information in this register is shown below:



The contents of this register must be maintained during an interrupt.

Next DW Addr in Prime–(GPR 9): Bits 8–31 maintain the address of the next input doubleword to be accessed from storage and are updated on a doubleword basis as each doubleword is fetched.

DBL-Word Count–(GPR 10): GPR 10 (bits 16–31) must initially contain the input doubleword count for the data to be processed by CVWL. This count includes both headers and data. CVWL decrements this count by one for each doubleword processed (either header or data).

Conversion Constant–(GPR 11): The conversion constant (bits 24–31) comprise a scaling control constant (bits 24–25) and a conversion multiplier (bits 26–31). The conversion constant is used to convert coordinates, relative to the PVD origin, from system scale to PVD scale. (The translation of system coordinates from system origin to PVD origin is performed during geographic filtering.)

Conversion constants may be derived from the following formulae:

$$M = \frac{4096}{S' \times 4^N} \quad (a)$$

$$S = \frac{4096}{M \times 4^N} \quad (b)$$

where S' is the nominal scale (PVD diameter in nautical miles).

S is the actual exact scale truncated to the nearest sixteenth of a mile.

M is the conversion multiplier and must be an integer in the range $1 \leq M \leq 42$.

N is the scaling control constant and must be an integer in the range $0 \leq N \leq 2$.

Given any nominal scale S', formula (a) may first be used to solve for any acceptable combination of M and N with M rounded to the nearest integer. For example, in solving for S'=28, N=0 would indicate M=146, which is invalid; for S'=28, N=1 would indicate M=37, which is acceptable; for S'=28, N=2 would indicate M=9, which is also acceptable but a poorer approximation.

The actual exact scale S should then be determined using the M and N values from formula (b). (Here, S must be truncated to the nearest sixteenth of a nautical mile, less than the actual quotient.) Continuing our example, in solving for M=9, N=2 (or M=36, N=1) would indicate S=28.44 (28 7/16<28.444); for M=37, N=1 would indicate S=27.63 (27 10/16<27.67). Note that in solving for M=16, N=0 would indicate S=255.94 (255 15/16<256.00).

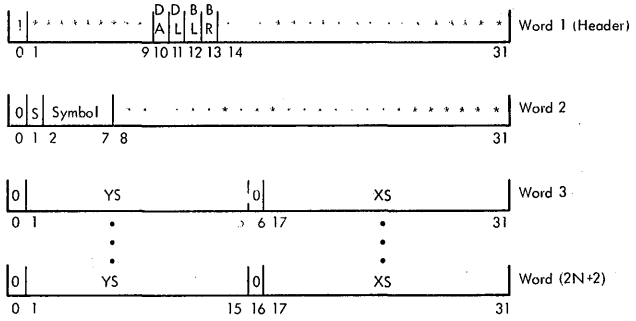
GPRs 12–15: These registers may be used by the programmer.

FPRs 0, 2, 4, 6: The floating point registers define the PVD geographic area and the three sterile areas in terms of 15-bit system coordinates given relative to the system origin.

The input data stream, shown below, consists of a header doubleword (symbol and display information), some number of data doublewords (X- and Y-coordinates defining both ends of a weather line), another header doubleword, more data doublewords, etc. The information in the

header pertains to all of the weather lines (data doublewords) that follow it; the number of doublewords is variable, depending on the weather input to the system.

Input Format



Legend:

DA	Dash
DL	Dash length
BL	Blink
BR	Brightness
S	Symbol Size
N	Number of weather lines
*	Ignored by CVWL

The symbol and display information (from the header) appears in each output doubleword, along with two sets of coordinates converted to the PVD's scale. One of these sets of coordinates (Y1, X1) defines one end of the line and is referred to as the major position. The other set of coordinates (ΔY2, ΔX2) defines a secondary point relative to the major position, rather than another unique point on the face of the PVD. If one end of the weather line is within the PVD's boundaries and the other end is outside, truncation is required, and the end that gets truncated becomes the second point.

After a weather line passes the filters (is within the PVD geographical area and outside the sterile areas) it must be formatted for the output doubleword. Since the system coordinates represent nautical miles, they must be converted to display units (relative to the PVD origin) according to the PVD scale selected. If the selected scale is 128 nautical miles per display diameter, each display unit is 128/1024=1/8th nautical mile. The conversion is accomplished by multiplying the coordinates by the proper conversion constant.

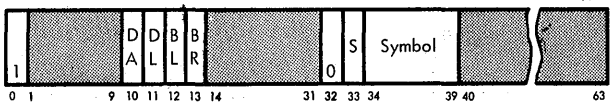
Next, the delta values (ΔY2, ΔX2) are calculated. Subtraction of the major position from the secondary position provides these values and a sign for each. The sign indicates the direction from the major position to the secondary position.

Three format words, named for their relationship to the three format micro-orders (FMTW-0, FMTW-1, FMTW-2), are assembled from the processed information and put into M to be gated, through the mixer, into XY in output doubleword format. After this doubleword is stored in the PVD's refresh memory, and if there are no pending interrupts, the next input doubleword is read from the input stream.

Interrupts can be honored only after each input doubleword has been completely processed. Since the control information is updated as it is used, processing after re-entry to the CVWL instruction begins at the point where it was interrupted.

Because CVWL execution begins by reading local storage per the R2 field, to get the beginning address of the input stream from GPR 9, the R2 field must always contain 9. The input stream address is gated to D, and a memory request is made (per D). The address of the next instruction (IC) is saved by gating it into the LSWR. This allows IC to be used for holding and decrementing the word count.

The first doubleword read from the input stream into AB should contain header information. A header doubleword is identified by the value of bit 0 (1 indicates a header, 0 indicates a data doubleword).

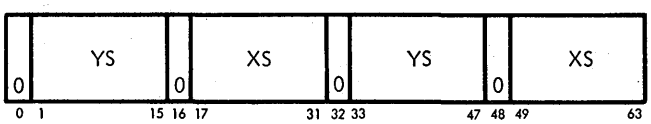


When the input doubleword is read from memory, bit 0 is tested. The outcome of this test determines which routine is to be entered.

If the input doubleword is a header, its information must be saved for later use in building the output doublewords. To accomplish this, the control bits (in A 8–15) and the symbol (in B 32–39) are combined into one word and stored in GPR 8. Then, the next input doubleword is read from the input stream into AB, and its bit 0 is subjected to the same test. This doubleword should contain data; if so, bit zero being 0 will cause the previous routine to be bypassed and the routine that processes the data to be entered. If the input stream should contain another header, the old header information in GPR 8 is replaced by the new header information.

The data count (furnished by software and in GPR 10 bits 16–31) is gated into IC and is decremented for each doubleword processed. When the count decrements to 0, all of the input stream has been processed, and CVWL terminates its execution. If the count does not equal 0 after it is decremented, it is stored back in GPR 10.

The data doubleword contains two sets of system coordinates, which define both end points of a weather line.



One set of coordinates is processed at a time, and the first set to be processed (word 1) is in A. The second set (word 2 in B) is gated to K. STAT D is turned on to indicate to the microprogram that word 2 has been placed in K and still needs to be processed.

The first check performed on word 1 is the geographic filter. First, it is determined whether the end of the weather line defined by word 1 is within the geographic area of this PVD. Two sets of coordinates, which describe the geographic area of the PVD, are supplied by software and are in FPR 0. For the discussion of the geographic filter, refer to the introduction to the 9020 display instructions under the heading "Introduction to CSS and CVWL".

If word 1 fails the filter, it is saved in GPR 0 and CVWL proceeds to test word 2. (If the other end of the weather line passes all of the filters (geographic and sterile areas), word 1 coordinates will be truncated.) Word 2 is gated from K to A, STAT D is reset, and the geographic filter is re-entered to begin processing word 2.

If the limit latch is still off after the geographic filter, the input point is within the PVD's geographical area. A check is now made to see if the point is in any of the three sterile areas. Each of the sterile areas is also defined by two sets of coordinates. These were furnished by software and are obtained by CVWL from FPRs (sterile area 1 from FPR 2, sterile area 2 from FPR 4, and sterile area 3 from FPR 6). Failure of any of the sterile area filters always causes the line to be rejected (i.e., the microprogram branches back to the beginning, a new input doubleword is read in, and CVWL begins processing it). See "Introduction to CSS and CVWL" for a discussion of the geographic and sterile area filters.

If the first word processed passes the geographic filter and the sterile area filters, it is saved in GPR 0. The result obtained from the first subtraction in the geographic filter (input coordinates minus origin coordinates) is also saved (in GPR 1). This value is known as the major position and, after being converted to the PVD's scale, appears in the output doubleword. If the second set of coordinates passes all of the filters, the result of the subtraction for word 2 is also saved. This result is known as the secondary position and is used to compute the delta values (which are the required input to a PVD and appear in the output doubleword).

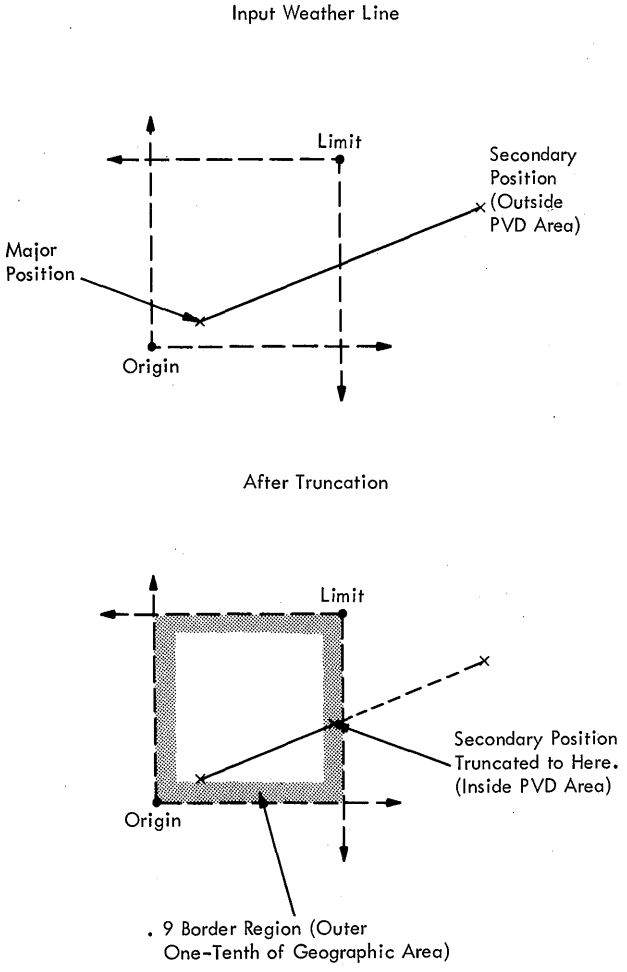
After both words of an input doubleword have been through the filters and the line has not been rejected (at least one word has passed all filters), two possible conditions exist. These conditions determine which routine must be entered next.

If both sets of coordinates for the weather line pass all of the filters, truncation is not required. Both ends of the weather line are already within the PVD's geographical area. The truncation routine is bypassed, and the routine that converts the coordinates to the scale of the PVD is entered directly.

The truncation routine is entered if one set of the input coordinates failed the geographic filter and the other set passed all filters (and was designated as the major position). One end of this input weather line is in the PVD's geographical area, but the other is not. The line is shortened so that the secondary position lies within the 0.9D border region of the PVD's area. This can be seen in the illustration below.

The truncation routine is entered, with the major position in T and the secondary position in A. T and A are added together, and the result is divided by two and gated to B. The truncated point obtained (B) is now subjected to the geographic filter again, using the limit latch on (as before) as an indication of failure. Failure causes B (the truncated point) to be gated to A and the major position (in GPR 0) to be gated back into T. The microprogram then branches back to the beginning of the routine; another truncation is performed, and its result is checked for geographic filter failure.

When the truncated point passes the geographic filter, the 0.9D border region coordinates (origin in FPR 6 and limit in FPR 7) are gated into T, one at a time, and the border filter is performed. These coordinates define the inner boundary of the 0.9D border region; if the point fails this filter (limit latch on), it is in the desired area, i.e., inside the PVD geographic area but outside the inner boundary of the 0.9D border region.



When that occurs, truncation is complete, and the conversion routine is entered. Should the point pass the 0.9D border filter (limit latch off), this indicates that the last truncation shortened the line too much. Since the secondary position must fall within the 0.9D border region, the truncated point must now be moved outward. The truncated point (B) is gated to T in place of the major position; since the secondary position, which was the last point to fail the geographic filter, is still in A, the same truncation routine can be entered again, and outward truncation takes place. A maximum of nine truncations are performed, with F being used as a counter (set to 0A, decremented, and checked after each truncation). If F reaches 0 before the end of the weather line has been moved into the 0.9D border region, the line is rejected, and the microprogram branches back to the beginning to read in a new input doubleword.

After truncation is complete and the truncated point lies in the 0.9D border area of the PVD, the secondary position must be calculated. This is done as it would have been if truncation had not been required and if entry to this routine had been directly from the filter routines. The PVD's geographic origin coordinates are gated into T (from FPR 0) and subtracted from the truncated point (moved to A). The result (secondary position) is stored in GPR 1, and the routine to convert the coordinates to the PVD's scale is entered.

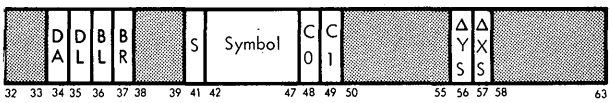
STAT D is turned on before entry into the conversion routine, causing control to be given back to CVWL after each conversion (CSS also uses this routine). The routine converts one set of coordinates at a time; therefore, it must be entered twice to process a complete doubleword, with the major position being converted first.

The coordinates to be converted are in B, and the conversion constant is in S, no matter where entry is from. B is multiplied by S, using the 'SEL-MPL * E3' micro-order; the result is a 10-bit Y-coordinate and a 10-bit X-coordinate. STAT D on returns control to CVWL, where the converted major position coordinates are stored in GPR 2 (X-coordinate) and GPR 3 (Y-coordinate). Before branching back to the conversion routine, CVWL gates the secondary position coordinates from GPR 1 into B and the conversion constant (GPR 11) back into S. STAT H is turned on to indicate to CVWL, when it regains control, that both the major and the secondary position coordinates have been converted.

Re-entry to CVWL after the second conversion is directly into the routine that calculates the delta position. The delta position is the difference between the major position and the secondary position (the length of the line). Since only one position and a length are provided to the PVD, a direction must also be given. This is obtained during the calculation of the delta value, which subtracts T (major position) from B (secondary position). The result of the subtraction is the delta value. F (bit one for delta X, bit 0 for delta Y) is set to the sign of the result (1 if the result is negative). These bits will become the ΔXS and ΔYS bits in the output doubleword. (The ΔXS, ΔYS bits being 1's indicates that the PVD must subtract the delta coordinates from the major position coordinates. The PVD adds the delta coordinates to the major position coordinates if the ΔXS, ΔYS bits are 0.)

The delta X-coordinate and ΔXS bit are computed first; delta Y-coordinate and ΔYS bit are computed next. If either delta coordinate is negative, it is given in two's complement form.

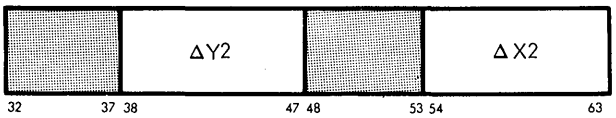
The delta X-coordinates are gated to B, and delta Y to A; they are saved there until format word 2 is assembled in T. The ΔXS and ΔYS bits are gated through the mixer into XY as part of format word 0.



The header information, saved previously in GPR 8, is now read into T to begin the assembly of format word 0. At this time, the ΔXS, ΔYS bits (F 0–1) are gated into T(56–57).

Two more bits required for format word 0 are the C0 and C1 bits. C0 equal to 1 indicates (to the PVD) that the symbol is to be displayed at the major position of the weather line. C1 equal to 1 indicates the same thing concerning the secondary position of the weather line. The symbol is normally displayed at the major position (except for an all-0's character). No symbol is displayed at the secondary position if the line was truncated or the symbol is all-0's. When a failure of the geographic filter occurs, the constant 2 is set into GPR 4. GPR 4 is now read into S and gated to PAL. PAL equal to 0 indicates that no truncation was performed, and the symbol can be displayed at both ends of the weather line. F (0 and 1) are both set to 1 (bit 0 is C0, and bit 1 is C1). If PAL does not equal 0, only F bit 0 is set to 1. The symbol is checked for a non-0 character by gating T(40–47) to the serial adder. An all-0 SAL now causes F to be reset and no symbol to be displayed at either end of the line. If SAL is not all 0's, the C0 and C1 bits are placed in T(48–49) by gating F(0–7) to T(48–55). Assembly of format word 0 is now complete, and T is gated to M.

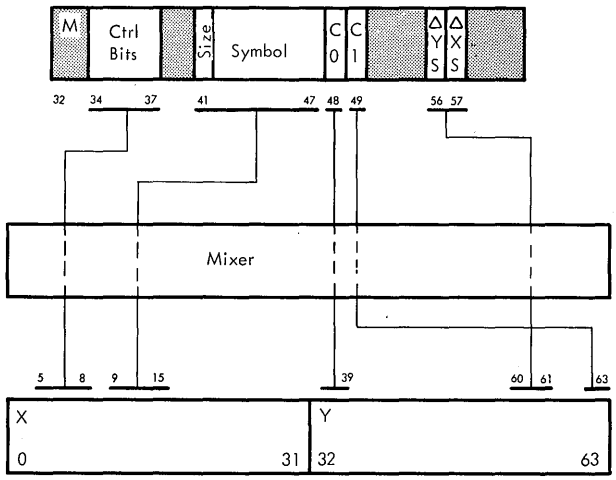
Format word 2 is assembled next in T. Assembly is started by gating B (ΔX) to T(48–63).



ΔY is in A(0–15) and is moved to T by gating A(0–7) through the serial adder into T(32–39) and then A(8–15) through the adder into T(40–47).

M (format word 0) must now be set into XY. This is done by issuing the 'FMTW* E14–15' micro-order. E14–15 being equal to 0 gates format word 0 (M) through the mixer and into the correct XY bit positions for the output doubleword.

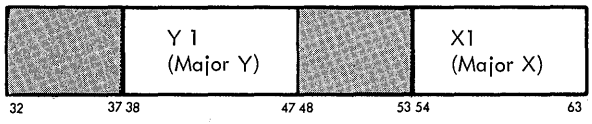
"FMTW*E 14-15" (E14-15 = 0)



This micro-order also causes XY bits 0, 1, 36, 37, and 62 to be set to 0's and XY bits 2, 3, 4, and 38 to be set to 1's.

Format word 2 (in T) is now gated to M, and format word 1 is assembled in T.

The major position coordinates are the contents of format word 1.

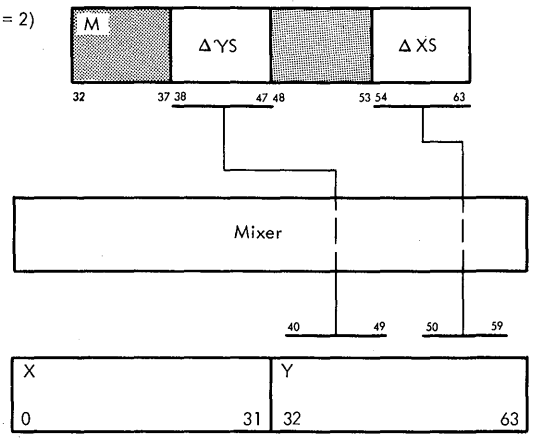


The major X-coordinate (in GPR 2 16–31) is read into T(48–63). The major Y-coordinates is in GPR 3 (16–31) and must be gated to T(32–47). This is accomplished by gating GPR 3 to A (via S and then B). A(16–23) is then gated through the serial adder to T(32–39); then, A(24–31) is gated through the serial adder to T(40–47).

Format word 2 (M) must be gated through the mixer and into XY. E(14–15) has been incremented to 2 since the last 'FMTW *E14–15' micro-order was issued. 'FMTW * E14–15' is issued now, and it gates the delta values, through the mixer, into the proper XY positions for the output doubleword.

"FMTW * E14-15"

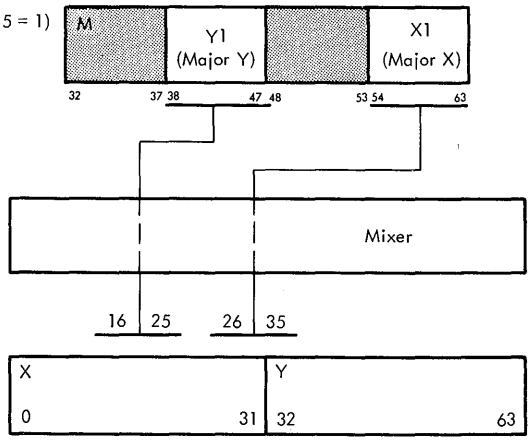
(E14-15 = 2)



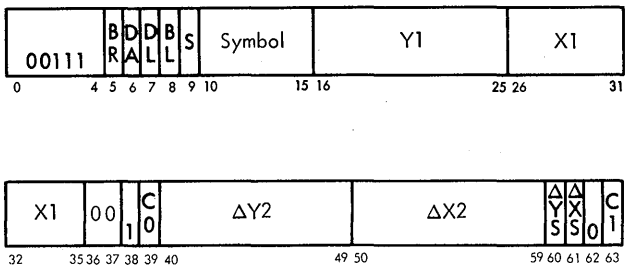
Format word 1 (T) is gated into M, and the refresh memory address (GPR 5) is gated into D (via S). Then, the address is updated (+8) and stored back into GPR 5. Gating format word 1 (M) through the mixer completes the assembly of the output doubleword in XY. E(14–15) has been decremented to equal 1, and the 'FMTW * E14–15' micro-order is issued again.

"FMTW * E 14-15"

(E 14-15 = 1)



The output doubleword is stored in refresh memory (per D) to complete the processing of one input doubleword.

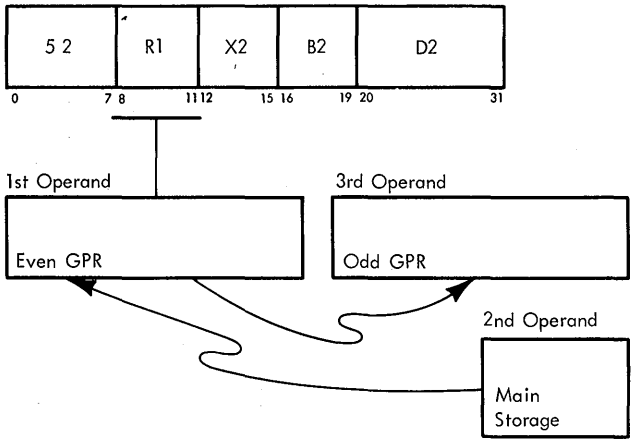


Any pending interrupts are now honored, so the address of the next instruction (LSWR) is gated back into IC (via S). If a nonprogram interrupt is pending, IC is decremented to point to the CVWL instruction; if a program interrupt, IC is left pointing to the address of the next instruction. After a nonprogram interrupt, control is given back to CVWL to continue processing the input stream from where it was interrupted.

If no interrupts are pending, the next input doubleword is read into AB, and the microprogram branches back to the beginning routine to check for header information.

LOAD CHAIN, LC (52)

- Moves first operand to third operand location and second operand to first operand location.
- R1 field specifies an even/odd pair of GPRs. Even GPR contains operand 1; odd GPR is the third location.
- RX Format:



- Condition Code:
  0–If bit 31 of second operand is 0.
  1–Not used
  2–Not used
  3–If bit 31 of second operand is 1.
- Program Interruptions

  Protection (fetch protect violation).
  Addressing (second operand is not a valid address).
  Specification (second operand address is not on a doubleword boundary).
- LC is useful in situations that require management of core storage (i.e., in accessing chained or linked blocks of data).
- Refer to Diagram 5-906, FEMDM.

At the beginning of execution, the contents of the GPR specified by the R1 field are in T, and a main storage request has been made for the second operand. The first operand (T) is stored in the odd register of the even/odd pair by setting the local storage address register to the value in R1 plus one.

After a delay to allow the data to arrive from memory, the second operand is set into ST. D(bit 21) (the second operand address) is checked; if it is a 0, the second operand is in S and must be moved to T so that it can be stored in the GPR. This is done by gating S through the parallel adder and into T. If the second operand is in T to begin with (D bit 21 = 1), this step is bypassed. T is then written into the GPR specified by E(8–11) (the R_1 field).

The instruction's objectives have been accomplished, except for the setting of the condition code. T(bit 63) is checked, and, if it is a 1, the condition code is set to 3. It is set to 0 if T(63) is a 0. The instruction execution is then terminated.

Maintenance aids available to maintenance personnel for the CE fall into two categories: (1) those used for error detection during normal operation, such as error-detection logic and interruptions, and (2) those used for diagnosing the cause of failures and for preventive maintenance. The second category includes:

1. CE control panel: Contains the controls necessary for initiating any operation, for manual testing, and for performing various maintenance tasks. In addition, indicators allow monitoring of the CE operation by displaying the status of important registers and control triggers.
2. Maintenance panel: Contains the controls necessary for initiating power on and off sequences. In addition, indicators show the power status.
3. Diagnose instruction: Allows certain diagnostic functions to be performed on the CE. It is used in conjunction with the maintenance control word (MCW) to allow such diagnostic functions as reversing parity and suppressing data checks. In addition, it can be used to initiate the logout and FLT functions.
4. MCW: Used in conjunction with the Diagnose instruction, FLTs, and ROS tests.
5. Logout, ROS tests, and FLTs: Logout stores the status of the console indicators into fixed positions for main storage when a trouble occurs; the data logged out may be subsequently recalled for analysis. ROS tests check each bit position of every ROS word. FLTs check the CE at the logic-block level.
6. Ripple tests: Provide the capability of (1) storing data from the DATA switches into all addresses in LS or main storage and inserting 1's with correct parity into the storage-protect keys and (2) reading out all locations of LS or main storage and displaying the data.
7. Diagnostic programs: Check the CE on a functional basis by programming it to perform one or more instructions or sets of instructions.
8. Marginal checking: Allows operation of critical circuits with nonstandard voltages to detect whether any are approaching failure.

The maintenance aids have interrelated functions, dependent upon the troubleshooting technique used. For example, scan logic, which provides the controls necessary to perform ROS tests and FLTs, is also used in logout, a CE control panel function. On the other hand, switches on the CE control panel are used to initiate all CE maintenance programs (i.e., FLTs, ROS tests). CE diagnostics may be initiated at either the SC/CC or CE control panels.

This chapter is divided into two sections: Section 1, Manual Controls, discusses the manual controls and indicators on the CE control panel and their application. Section 2, Maintenance Facilities, discusses the Diagnose instruction, MCWs, logout, ROS tests, FLTs, ripple tests, diagnostic programs, and marginal checking.

SECTION 1. MANUAL CONTROLS

CE CONTROL PANEL SWITCHES AND FUNCTIONS

The CE control panel, in addition to its main function as the operating and monitoring center of a 9020 subsystem, is one of the prime maintenance aids available to maintenance personnel. Using this panel, maintenance personnel can duplicate many program operations or portions of operations manually and can repeatedly exercise portions of the machine logic at a normal or an increased rate of operation.

In the following discussion, coordinates are given [e.g., SYSTEM INTERLOCK switch (N47)] to assist in locating switches and indicators in Diagrams 6-1 and 6-2, 7201-02 CE FEMDM. Another useful reference for this section is Diagram 6-3, FEMDM, which, together with Table 4-1, will aid in understanding the environment in which the various functions of the switches and pushbuttons are active.

Table 4-1. CE Switches and Their Operational Environment

Switch	CE Program State		Controls at CE										
	Operating	Stopped	Key Off State					Key On** State					
			3	2	1	0	ø	3	2	1	0	ø
Address Compare	X				X	X	X	X	X	X	X	X
Address *			X	X	X	X	X	X	X	X	X	X
Backspace FLT	X	X				X	X				X	X
Check Control	X	X				X	X				X	X
Check Reset	X	X				X	X				X	X
Data *			X	X	X	X	X	X	X	X	X	X
Defeat Interleaving	X					X	X				X	X
Disable Interval Timer	X					X	X				X	X
Display		X			X	X	X	X	X	X	X	X
Element MPO	X	X	X	X	X	X	X	X	X	X	X	X
Frequency Alteration	X	X					X					X
Indicate On Roller 1, Pos 6	X	X	X	X	X	X	X	X	X	X	X	X
Inhibit CE Hard Stop	X	X				X	X				X	X
Interrupt	X				X	X	X	X	X	X	X	X
Key (System Interlock)	X	X	X	X	X	X	X	X	X	X	X	X
Lamp Test/Allow Indicate	X	X	X	X	X	X	X	X	X	X	X	X
Load ***	X	X			X	X	X	X	X	X	X	X
Load Unit *					X	X	X	X	X	X	X	X
Logout	X	X			X	X	X	X	X	X	X	X
Main Storage Select			X	X	X	X	X	X	X	X	X	X
Marginal Check and Voltage Controls	X	X	X	X	X	X	X	X	X	X	X	X
Power Off	X	X					X					X
PSW Restart ***	X	X			X	X	X	X	X	X	X	X
Pulse Mode	X	X					X					X
Rate		X			X	X	X	X	X	X	X	X
Register Select *		X					X					X
Register Set		X					X					X
Repeat Instruction		X			X	X	X			X	X	X
Reset	X	X			X	X	X			X	X	X
ROS Address	X	X			X	X	X			X	X	X
ROS Transfer	X	X			X	X	X			X	X	X
Scan Mode	X	X				X	X				X	X
Set IC		X			X	X	X	X	X	X	X	X
Start		X			X	X	X	X	X	X	X	X
Store		X			X	X	X	X	X	X	X	X
Stop	X				X	X	X	X	X	X	X	X
Storage Select *					X	X	X	X	X	X	X	X
Test Switch	X	X				X	X				X	X
1052 Enable-Disable	X	X	X	X	X	X	X	X	X	X	X	X
360 Mode	X	X			X	X	X			X	X	X

Legend:

X – Control is functional.

* – Static; requires use of other control.

** – If both CE and SC or CC keys are on, the SC or CC key takes precedence.

*** – System function with SYSTEM INTERLOCK key on, TEST switch off.
Subsystem function if SYSTEM INTERLOCK key off, regardless of TEST switch.

ø – State 0 and Test mode.

Some of the maintenance routines that can be performed from the CE control panel include storage ripple, marginal checking, and frequency bias. Other controls allow maintenance personnel to stop the CE at the end of the current instruction, to display main storage or LS, to store into main storage or LS, or to log out indicator status to fixed positions in main storage. (The latter is a function of the scan logic and is described in Section 2 of this chapter.)

SYSTEM INTERLOCK SWITCH

The SYSTEM INTERLOCK switch (N47) is a key-operated, lockable switch. With the key inserted and twisted clockwise, the SYSTEM INTERLOCK switch is turned on and enables manual controls as indicated in Table 4-1. Note that the CE SYSTEM INTERLOCK switch is overridden by the SC/CC SYSTEM INTERLOCK switch if the CE is selected by the SC/CC CE SELECT switch. The LOAD and PSW RESTART functions are changed from subsystem to system functions when the CE SYSTEM INTERLOCK switch is turned on. These are further discussed in the applicable portions of this chapter.

TEST SWITCH

The CE TEST switch (H7) is enabled only when the CE is in state zero. It is indicated both on the TEST indicator, and on roller 6 position 5 bit 5. The Test switch enables four manual CE Control Panel switches which are not otherwise enabled by state zero: REGISTER SELECT, REGISTER SET, PULSE MODE (time or count), and FREQUENCY ALTERATION. It prevents the CE from issuing SCON or SATR and prevents the following external signals from entering the CE: ELC interrupt, RDD, WDD, SATR, SCON, SC/CC START or STOP, and SYSTEM (EXT) reset.

SYSTEM RESET PUSHBUTTON

The SYSTEM RESET pushbutton is enabled in any state if the CE System Interlock key is on and the CE is not selected by the SC/CC with its key on. It performs Subsystem Reset. A Subsystem Reset will also be performed if the CE is in state one, zero, or test with the CE System Interlock Key off (System Reset which is in effect, External Reset, is a function of System IPL or System PSW Restart only). Refer to Chapter 9 of the 9020D or E System Introduction Manual for definition of system and subsystem resets as they affect other system elements.

A CE Machine Reset (reset within the CE) will result from any of the following: system or subsystem reset, system or subsystem IPL, system or subsystem PSW restart or an external start, external stop, or external reset.

A machine reset resulting from any of these functions forces ROSAR to 003 which is the stop loop. When the CE is powered on, the same machine reset occurs except that ROSAR is forced to 00B (POWER ON RESET). A microprogram which clears local storage is then executed before the stop loop is entered. Refer to FEMDM Diagrams 6-4 and 6-7.

STOP LOOP

- The stop loop is entered via three hardware-forced ROS words: 003, 00B, and 026.
- The MANUAL indicator is on when the CE is in the stop loop.
- The stop loop is exited by branching into one of eight pushbutton function microprograms or into one of four external condition microprograms.

When the CE is in the Stopped state, a ROS microprogram called the "stop loop" is continuously executed. The stop loop is shown in Diagram 6-4, FEMDM.

When the CE is in the stop loop, the MANUAL indicator is lit, no program instructions are executed, the interval timer (location 50, hex of the PSA) is not stepped, and the starting location of the next instruction to be executed is transferred to D (contents of IC minus 8 or 16). D is displayed in roller 1, position 2; IC is displayed in roller 6, position 3.

Stop Loop Entries (Diagram 6-4; FEMDM): The stop loop is entered via any of three ROS words the addresses of which are hardware-forced: (1) 003, (2) 00B, and (3) 026. Each of these ROS words issues the 'set stop loop trigger' micro-order, which turns on the MANUAL trigger. This trigger [plus state and test switch considerations (Table 4-1)] enables manual operations except SYSTEM RESET, CHECK RESET, and LOAD pushbutton functions. These are active regardless of the stop loop. (1) ROS word 003 is forced by system or subsystem: PSW restart, IPL, Reset, or by EXTERNAL: Start, Stop, or Reset. The stop loop is entered directly. (2) The STOP trigger is not turned on, except for External stop, and ROS word 00B is forced by a power-on reset. Local storage is cleared before the stop loop is entered. The STOP trigger is not turned on. (3) ROS word 026 is forced when the STOP trigger is on at end op. When all pending interruptions and I/O operations are completed, the STOP trigger forces address 026 into ROSAR to enter the count delay microprogram which allows time to recognize that a pushbutton has been depressed before the stop loop is entered. The STOP trigger

is turned on by the STOP pushbutton, an address compare with ADDRESS COMPARE STOP on, RATE switch in instruction step position, External stop, or the 'logout PB latch being on after logout (scan-out) is completed.

Stop Loop Exits: The stop loop continuously samples eight CE control panel pushbuttons and four external conditions. Each of these will cause a branch from the stop loop into a new microprogram to perform the indicated function (Diagrams 6-4 and 6-5, FEMDM). CE control panel pushbuttons sampled are STORE, DISPLAY, SET IC, START, ROS TRANSFER, REG SET, IPL (system or subsystem), and PSW RESTART (system or subsystem). The external conditions tested are External Start and ATR Select.

STOP PUSHBUTTON

The STOP pushbutton causes entry to the stop loop (see "stop loop") while retaining machine environment (Diagram 6-6, FEMDM). It is enabled in state one, zero, or test or in any state with the System Interlock key on unless the CE is selected by the SC or CC with its System Interlock key on. The operator may perform manual operations (store, display, etc.), acknowledging state and test switch considerations (Table 4-1) or continue normal program operation by depressing START.

CHECK RESET PUSHBUTTON

The CHECK RESET pushbutton or FLT check reset (an FLT test function) performs the error reset portion of machine reset. The CHECK RESET pushbutton is enabled in state zero or test. It does not change the mode of operation which continues as if no error had occurred. Results are unpredictable. Scan machine reset (a scan-mode microprogram function) also performs a partial machine reset. None of these resets forces the CE into the stop loop. Refer to Diagram 6-7, FEMDM.

To reset a hard-stop error, actuate the INHIBIT CE HARD STOP switch (then, if it is a PSBAR parity check, set the correct parity in PSBAR), and depress CHECK RESET.

START PUSHBUTTON

The START pushbutton starts the CE in any Rate mode (RATE switch setting: Process, Single Cycle, Instruction Step). Diagram 6-4, FEMDM, assumes the rate switch to be selecting PROCESS or INSTRUCTION STEP. The START pushbutton is active only in state one, zero, or test unless the CE System Interlock key is on. It resets the STOP and MANUAL triggers and sets the (block) 'interrupts' latch (Diagram 6-6), FEMDM). When the START pushbutton is depressed, an end-op occurs and processing begins with I-fetch of the next instruction. The 'interrupts' latch blocks interrupts until the end-op of the first instruction. If START is depressed after a normal halt, instruction processing continues as though no halt had occurred, and pending interruptions are taken after execution of the first instruction. If START is depressed after a system reset or a write direct stop, the results are unpredictable.

MAIN STORAGE SELECT AND LOAD UNIT SWITCHES

The MAIN STORAGE SELECT and LOAD UNIT switches (R45–55) are used exclusively by the IPL and PSW Restart functions (Diagram 6-8, FEMDM). They are the operator's means of specifying which SE contains the Restart PSW at PSW location 0 (for PSW Restart) or the SE into which the initial program is to be loaded and by what channel and I/O unit (for IPL).

LOAD PUSHBUTTON

The LOAD pushbutton (S-52) is used to initiate an Initial Program Load. It is enabled at the CE control panel only in state one, zero, or test if the CE System Interlock key is off. It is enabled in any state with the CE key on unless the CC/SC with System Interlock key on has the CE selected.

IPL

- There are three types of IPL from the CE: System, Subsystem, and ROS test or FLT.
- IPL from the CE causes the IOCE to load a program from an I/O unit. The program is then executed by the CE.

There are three types of initial Program Load: system IPL, subsystem IPL, and FLT or ROS IPL. All three are alike in that the CE raises the line IPL IOCE (X) to initiate the I/O operation. The major differences are that the CE sends SCON and SATR in a system IPL, only SATR in a subsystem IPL, and neither in an FLT or a ROS IPL. The system and subsystem IPL are discussed below. For a discussion of FLT and ROS test IPL, refer to Section 2 of this chapter.

System IPL

A system IPL can be initiated from the SC/CC or a CE with the CE, SE, unit, and channel select switches and the System Interlock key set (Diagram 6-8A, FEMDM). The selected CE issues a system reset to all elements turning on all SCON bits and then takes the information from the

SC/CC or its own load unit select switches to configure a subsystem: the CE places the selected SE number in its own ATR position 1 and physical PSBAR, issues a special IPL SCON to each of the selected elements, issues a SATR to the selected IOCE, and raises 'IPL IOCE (X)' to the IOCE.

The IOCE reads 24 bytes into the PSA in the selected SE, starting at location 0 as follows:

Location 0. IPL PSW 0 is read in. This is the first data that will be fetched by the CE. It includes the starting address of the program being read in by the IOCE for the CE.

Location 8. CCW 1 is read in. This is the first CCW to be fetched by the IOCE. It specifies the location and the number of bytes that the IOCE will read in. It may or may not specify chaining to CCW 2.

Location 10 (hex). IPL CCW 2 is read in. This CCW may be a transfer in channel (TIC) to another CCW or a data address and byte count with or without chaining specified.

Having read these three doublewords into the PSA, the IOCE fetches from location 8 and executes IPL CCW 1.

A bootstrap-type operation may then be performed in which IPL CCW 1 may cause further CCWs to be read into storage from the input device and chain to IPL CCW 2. CCW 2 would then specify a TIC to one of the CCWs just read in. The IOCE continues fetching and executing CCWs until one which specifies neither TIC nor chaining is fetched. It executes this CCW, then raises 'response' to the CE. This releases the CE to fetch IPL PSW 0 and start executing the program read in by the IOCE.

Subsystem IPL

A subsystem IPL can be initiated from either the SC/CC or a CE with the subsystem manually configured and the CE, SE, unit, and channel select switches and the System Interlock key set (Diagram 6-8B, FEMDM). The selected CE issues a subsystem reset to the elements configured and issues a SATR and an IPL to the selected IOCE. All subsequent actions are the same as in a system IPL (Diagram 6-9, FEMDM).

DATA SWITCHES

The 64 DATA switches (K, L, 14–49) (Diagram 6-1, FEMDM) allow the operator to enter data manually into the system. They are alternately colored black and white in groups of four to facilitate entering hex data into the CE. Each switch is a two-position toggle switch, with the up position equalling a 0 and the down position equalling a 1. When the DATA switches are used, the selected micro-program first places 1's into ST. Then, if a DATA switch equals a 1, the corresponding bit in ST is unchanged (remains set); if a DATA switch equals a 0, however, the corresponding bit is reset to 0.

The DATA switches are used during the following manual operations (DATA switch gating is shown in Figure 4-1):

1. Store.
2. Storage ripple store.
3. Repeat instruction.
4. Pulse mode count function (switches 53–63).
5. Register set.
6. Loop on address compare (switches 40–63).

The DATA switches are used during the following programmed operations:

1. Diagnose, with MCW(8–19) = 5B7.
2. FLT test number search after a stop.

When data is entered, correct parity is automatically generated by the switches. If the switches are altered during an operation, such as Repeat instruction or storage ripple, an error will probably occur.

ADDRESS SWITCHES

The 24 ADDRESS switches (M, 14–49) allow the operator to manually select any address in ROS, LS, or main storage. They are alternately colored black and white in groups of four to facilitate entering hex addresses into the CE. Each is a two-position toggle switch, with the up position equalling a 0 and the down position equalling a 1. During manual operations, the selected micro-program places 1s into D. If an ADDRESS switch equals a 1, the corresponding bit in D is unchanged. If an ADDRESS switch equals a 0, however, the corresponding bit in D is reset.

The ADDRESS switches are used during the following manual operations (ADDRESS switch gating to D, ROSAR, and comparison circuits is shown in Figure 4-2):

1. Store.
2. Display.
3. Set IC.
4. ROS transfer.
5. ROS repeat address.
6. Main storage address-compare stop, sync, or loop.
7. ROS address-compare sync or stop.

To address main storage or LS, the ADDRESS switches are used with the STORAGE SELECT switch (N13). Address switches 8–28 select a doubleword in main storage (Figure 2-28). Bit 8 is always 0. Bit 9–12 select the ATR slot (which SE/DE). If bits 9–19 are all 0 (PSA access), the SE is determined by physical PSBAR and the PSA block

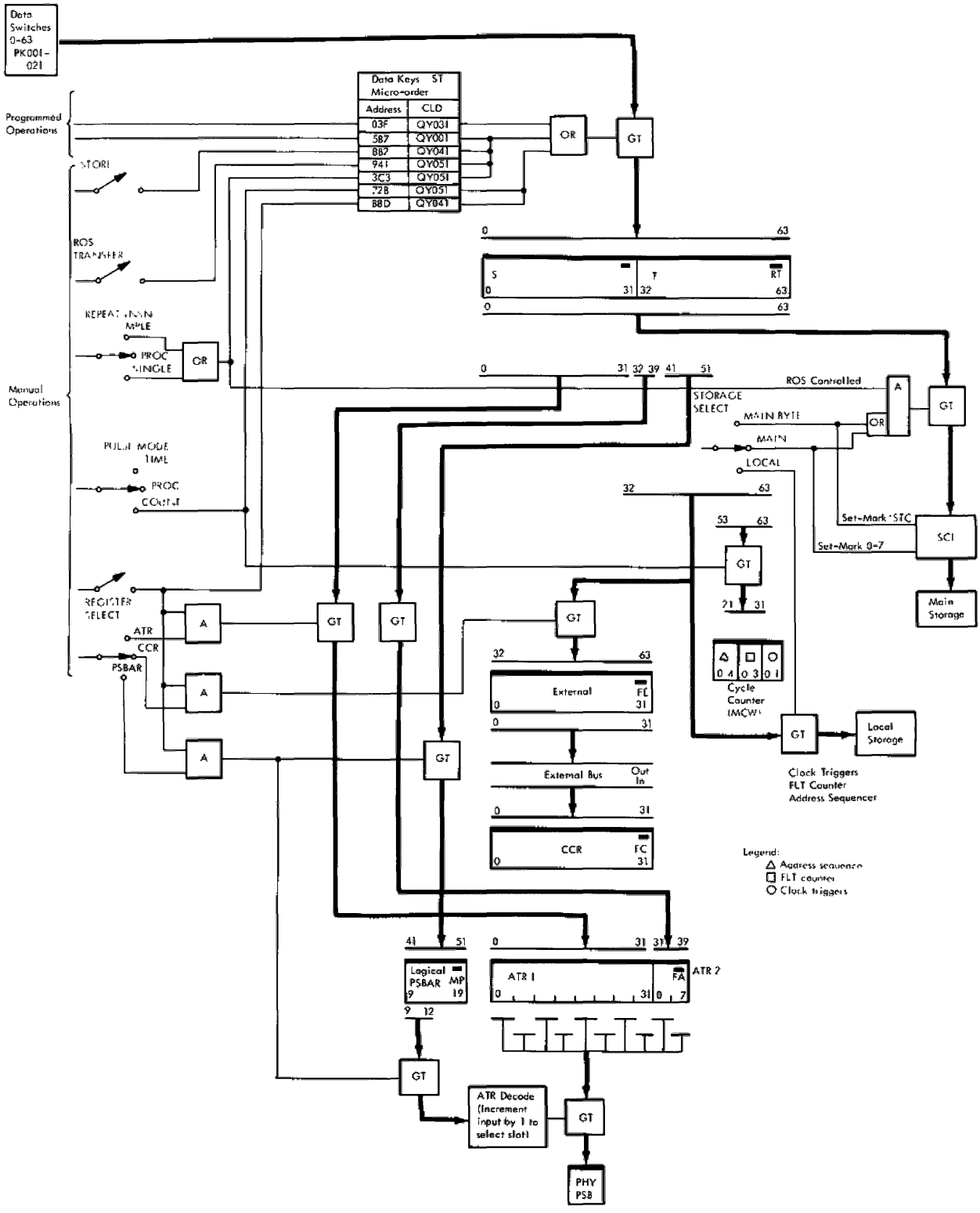


Figure 4-1. Data Switch Gating

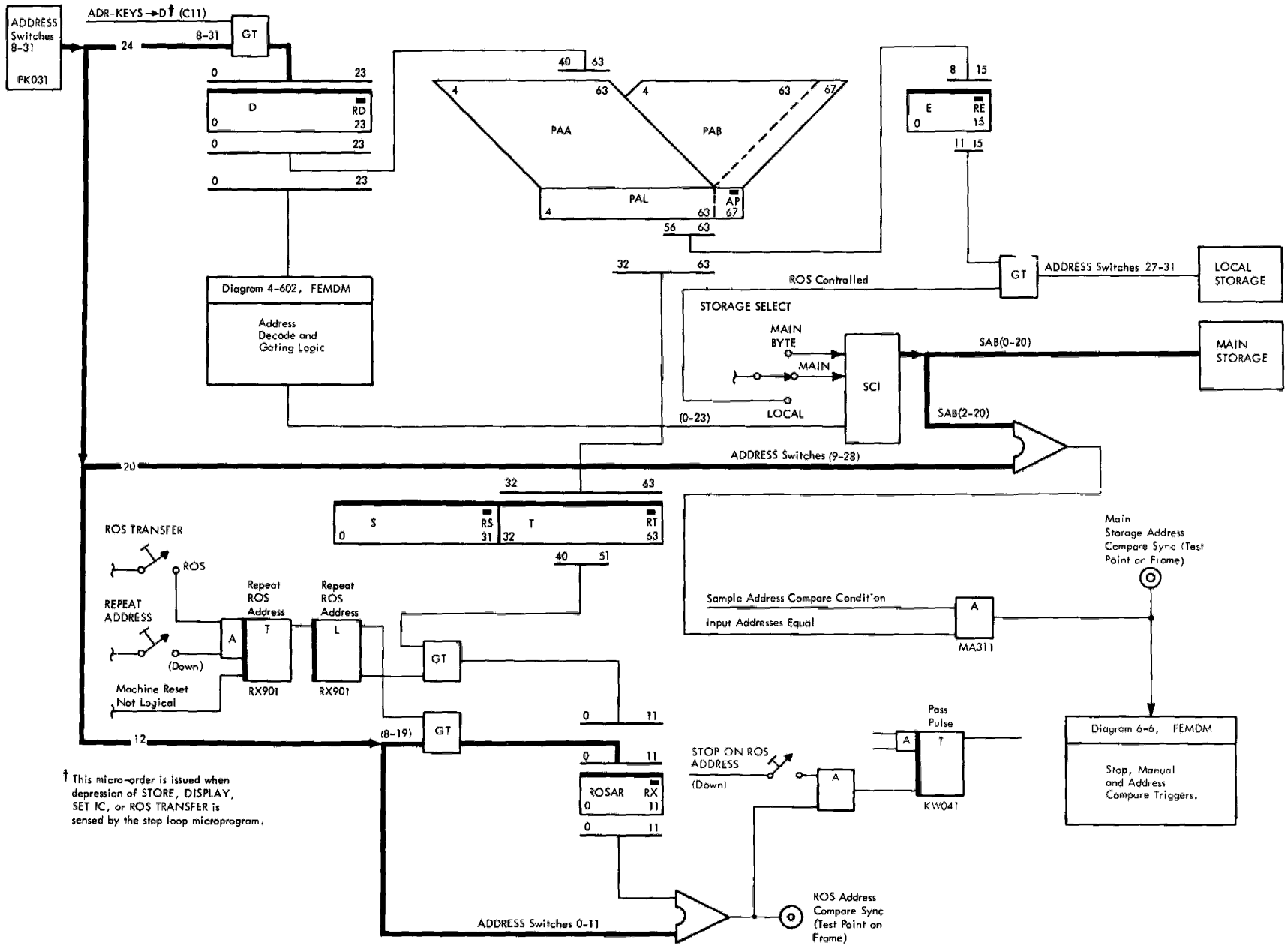


Figure 4-2. Address Switch Gating

location by bits 13–19 from logical PSBAR. ADDRESS switches 9–28 may be used in conjunction with the ADDRESS COMPARE STOP switch (N17) when selecting an address for an address-compare stop or sync. ADDRESS switches 27–31 select an LS address.

ADDRESS switches 8–19 select a ROS address to be used to obtain a ROS address sync (a sync pulse is generated when the ROS address and the ADDRESS switches agree) or to contain a ROS address for use with the ROS TRANSFER pushbutton (Q35).

When an address is entered, correct parity is automatically generated by the switches when gated to the CE.

ADDRESS COMPARE STOP/PROC/LOOP SWITCH

The ADDRESS COMPARE switch (N15) is enabled at the CE control panel in state one or zero or in test with the CE System Interlock key off. It is also enabled in any state with the CE System Interlock key on unless the CE is selected by the SC/CC with the Interlock key on. The ADDRESS COMPARE STOP/LOOP switch is duplicated at the SC/CC. Its function at either machine is the same.

STOP Position

In the STOP position, this switch enables the operator to stop the CE at a predetermined address by setting ADDRESS switches 9–28. When these switches match the address sent to the SCI, the 'address compare' trigger (Diagram 6-6) is set. (Address comparison is performed on doubleword boundaries.) The CE will stop at the end of the instruction in progress. The 'address compare' trigger causes the 'stop' trigger to be set, and the count delay microprogram and the stop loop are entered. This entry to the stop loop is identical with the entry that results from depressing the STOP pushbutton (Diagram 6-4, FEMDM).

PROC (Process) Position

In the PROC position, this switch allows normal CE operation.

LOOP Position

The ADDRESS COMPARE switch (LOOP position), in conjunction with ADDRESS switches and DATA switches, provides a means of manually forcing the CE to loop an entire instruction stream. The address of the last instruction of the stream to be executed is placed in ADDRESS switches 8–31; the address of the first, in DATA switches 40–63. The CE will proceed through the instruction stream until it gets an address compare (at the address in ADDRESS switches 8–31). The 'address compare' trigger is set, and it sets the 'stop' trigger. The execution of the instruction at the compare address is completed, and the CE enters the stop loop just as in an address compare/stop operation. However, the 'loop' trigger is also set, which causes the following (Diagram 6-6, FEMDM): The SET IC branch from the stop loop is taken, and that microprogram routine is executed (DATA switches 40–63, bits are set into IC); the CE returns to the stop loop. Then the START branch from the stop loop is taken, which puts the CE back into the instruction stream per IC (Diagram 6-4, FEMDM).

A synchronizing pulse which occurs with every address match (SAB 1–20 with ADDRESS switches 9–28) is available for maintenance scoping. A jackbox for this purpose is located on the machine frame, behind the CE control panel. This pulse is active regardless of the ADDRESS COMPARE switch setting (Diagram 6-6, FEMDM).

STORAGE SELECT SWITCH

The STORAGE SELECT switch (N13) is a three-position toggle switch that selects LS or main storage. It is used in conjunction with the ADDRESS switches and the STORE (R32) or DISPLAY (R35) pushbutton. The positions and corresponding functions are:

1. MAIN position – Selects the doubleword main storage location specified by ADDRESS switches 9–28 for storing or displaying data.
2. MAIN BYTE position – For storing a byte, selects the doubleword location (per ADDRESS switches 9–28) and byte (per ADDRESS switches 29–31) within the doubleword. For displaying, selects the doubleword in main storage to be displayed (same as MAIN).
3. LOCAL position – Selects an LS location (per ADDRESS switches 27–31) for storing or displaying data.

The STORAGE SELECT switch conditions hardware, so that a ROS branch may occur, by setting or inhibiting ROSAR(11). The 'SB-PB' and 'LS-PB' micro-orders allow the stop loop microprogram to perform the storage selection specified by the STORAGE SELECT switch. STORAGE SELECT switch gating is shown in Diagram 6-10, FEMDM.

The use of the STORAGE SELECT switch is discussed in the following paragraphs whenever it is involved in a specific manual operation.

DEFEAT INTERLEAVING SWITCH

The DEFEAT INTERLEAVING switch (N7) permits maintenance personnel to choose which halves of main storage are the high-order and low-order portions for maintenance use. It is a three-position switch that performs the following functions (Diagram 6-11, FEMDM) in state zero or in test only, regardless of the System Interlock key:

1. NO REV (up) position – Interleaving of main storage addresses is disabled, and locations in each BSM are addressed consecutively.
2. REV (down) position – Interleaving of main storage addresses is disabled, and locations in each BSM are addressed consecutively; the high and low BSMs are reversed.
3. PROC (center) position – Normal position of the switch. Addresses are interleaved in the normal manner with no reversal of storage addresses, unless changed by the Diagnose instruction.

By using this switch, maintenance personnel may address the first 16,384 doublewords consecutively rather than in an interleaved manner. When used for this purpose, and with the switch in NO REV, the first 16,384 doubleword addresses to be selected will be in the even BSM and the next 16,384 doubleword addresses will be in the odd BSM. In REV, the first 16,384 doubleword addresses will be in the odd BSM and the next 16,384 doubleword addresses will be in the even BSM.

The signals generated by this switch are sent to the SCI and to main storage. When the switch is in NO REV or REV, the TEST indicator (S48) is lit.

SET IC PUSHBUTTON

The SET IC pushbutton (R30) provides a means of entering an address into the current PSW. The pushbutton is active only in state one or zero or in test unless the CE System Interlock key is on. It sets the instruction-address portion (bits 40–63) of the PSW [D(0–23)] to the value specified by the ADDRESS switches. The CE is reset to the start of an I-fetch at that address. Four instruction halfwords are loaded into Q per the address in D (contents of the ADDRESS switches); 8 is then added to D, and the sum is placed in the IC.

The first addressed instruction halfword is loaded into R per D(21,22). If D(21,22) = 11, Q is loaded with the next group of four instruction halfwords per the IC, and 8 is added to the IC (Diagram 6-4, FEMDM). If D(21,22) $\neq$ 11, Q is loaded only once.

After the instruction halfwords are fetched and loaded into Q and R, the count delay microprogram is entered. After the count delay, the stop loop is re-entered. Further manual intervention is required to start program execution. [START (S25) is depressed.] Note that the CE must be in the Stopped state (stop loop) for this pushbutton to function. Note, too, that when the CE is in the Stopped state, the instruction address is contained in D. After the SET IC pushbutton is depressed, the new address contained in the IC is one or two doublewords more than the address contained in the ADDRESS switches.

RATE SWITCH

- Controls rate of instruction execution.
- CE must be in stop loop before switch is activated.

The RATE switch (Q25) selects the rate at which instructions are to be executed. It is active only in state one or zero in test unless the CE System Interlock key is on. This rotary switch has four positions: PROCESS, INSN (instruction) STEP, SINGLE CYCLE, and SINGLE CYCLE STORAGE INHIBIT:

1. INSN STEP (instruction step): CE executes one machine instruction for each time START is depressed.
2. PROCESS: Does not affect CE operation; CE operates at normal clock speed.
3. SINGLE CYCLE: CE advances by its minimum clock amount for each depression of START; all CE operations are the same as for the PROCESS position.
4. SINGLE CYCLE STORAGE INHIBIT: Same as SINGLE CYCLE without storage references.

Two latches and two triggers control the RATE switch operation: 'instruction step' and 'single cycle' latches and 'pass pulse' and 'block' triggers (Diagram 6-12, FEMDM).

The 'instruction step' latch performs two functions: (1) it allows setting the 'stop' trigger so that only one instruction is executed with each depression of START (S25); (2) it disables the stepping of the interval timer.

The 'single cycle' latch allows single-cycle operation. One machine cycle is allowed with each depression of START, unless the CE requests additional machine cycles. The interval timer is disabled by the 'single cycle' latch.

When the 'pass pulse' trigger is set, CE machine cycles are allowed. This trigger blocks the CE machine cycles when in the single-cycle mode (RATE switch in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT position).

The 'block' trigger is used in single-cycle operation to allow one clock signal to be gated to the CE each time START is depressed. The clock signal is blocked by resetting the 'pass pulse' trigger.

The TEST indicator is lit when the RATE switch is in any position other than PROCESS.

PROCESS Position

When the RATE switch is in PROCESS, the CE operates at the normal clock speed of 200 ns; this is the position for normal program execution.

INSN STEP Position

The INSN STEP position allows the execution of one machine instruction for each depression of START; any instruction may be executed. All I/O operations and interruptions (not masked off) are executed after the instruction is completed. The CE then re-enters the stop loop as though the STOP pushbutton had been depressed. The TEST indicator (S48) is lit when the RATE switch is in INSN STEP.

Instruction-step operation is shown in Diagram 6-13, FEMDM. The CE must be in the stop loop before entering or leaving the instruction-step mode; the interval timer is disabled in this mode.

When the RATE switch is moved to INSN STEP, the 'instruction step' latch is set if the CE is in the stop loop or if the 'pass pulse' trigger is reset (Diagram 6-12, FEMDM). The interval timer is disabled after the 'instruction step' latch is set. No further action occurs until START (S25) is depressed, at which time one instruction is executed. At end-op, the 'stop' trigger is set by an 'I-Fetch reset' micro-order or a 'reset interrupt triggers' micro-order, thus forcing the CE to enter the stop loop. The CE remains in the stop loop until further action is taken by the operator.

SINGLE CYCLE Position

The SINGLE CYCLE position allows the CE to advance one machine cycle (200 ns) each time START (S25) is depressed (Diagram 6-14, FEMDM). The CE must be in the stop loop or have the 'pass pulse' trigger off before entering the single-cycle mode; if in the stop loop, it remains there until START is depressed. The CE begins executing instructions one machine cycle at a time for each depression of START. In Diagram 6-14, it is assumed that no CE requests are generated; if an asynchronous device is used or if a storage request is given, however, more than one machine cycle is required. The single-cycle mode continues through all CE functions of the instruction to the point of initiation of the asynchronous operation. This operation begins on the next depression of START and runs to the completion point in a normal manner.

If the asynchronous device initiates an interruption request during single-cycle operation, the interruption is broken into single machine cycles. More than one depression of START is therefore required. The CE runs at normal machine speed in the stop loop.

When the RATE switch is in SINGLE CYCLE, the 'single cycle' latch is set (Diagram 6-12, FEMDM). If in the stop loop, the CE remains there until START is depressed, at which time it advances one machine cycle. The 'block' trigger is set as shown in Diagram 6-12. If there is no 'BCU hold on clock' signal, the 'pass pulse' trigger is reset after the 'block' trigger is set, thus inhibiting CE clock signal distribution. START must be depressed for each CE machine cycle advance. The TEST indicator (S48) is lit when the RATE switch is in SINGLE CYCLE.

SINGLE CYCLE STORAGE INHIBIT Position

The SINGLE CYCLE STORAGE INHIBIT position allows the CE to advance one machine cycle (200 ns) each time START is depressed. All CE requests are ignored, and asynchronous operations are suppressed.

Except for the inhibit signals sent to the SCI and inhibiting the 'stop 1' and 'stop 2' triggers from affecting the CE clock, the single-cycle-storage-inhibit function is the same as the single-cycle function, Diagrams 6-12 and 6-14.

REPEAT INSN SWITCH

- REPEAT INSN switch allows repetitive execution of one or up to four instructions.

The REPEAT INSN (instruction) switch (N23) provides a means of repeating a single instruction or of repeating up to four instruction halfwords. It is active in state one or zero or in test only, regardless of the CE System Interlock key. The REPEAT INSN switch has three positions (Diagram 6-15, FEMDM): PROC, normal CE operation; SINGLE; and MPLE (multiple).

A trigger-latch combination controls the repeat-instruction functions: 'repeat instruction adjust' trigger and 'repeat instruction initialization' latch.

The 'repeat instruction adjust' trigger forces a branch to the manual control repeat exceptional condition micro-program (ROS address 028, hex) at end-op of the start microprogram and sets the 'repeat instruction initialization' latch. STAT G is set to block re-entering the repeat-instruction microprogram at microprogram end-op. The trigger is set when the CE is in the Stopped state and the REPEAT INSN switch is in SINGLE or MPLE; it is reset when the CE is in the Stopped state and the REPEAT INSN switch is placed in PROC.

The 'repeat instruction initialization' latch blocks transfer to Q and stepping of the interval timer when in repeat-single-instruction mode. The latch is reset when in the stop loop, and the RATE switch (Q25) is placed in PROCESS. The CE must be in the Stopped state before

entering or leaving the repeat-instruction mode. See Diagram 6-16, FEMDM, for the repeat-instruction operations.

The TEST indicator (S48) is lit when the REPEAT INSN switch is in any position other then PROC.

Repeat Single Instruction. When the REPEAT INSN switch is in SINGLE, one instruction is continuously executed. The instruction to be repeated is entered into the DATA switches, beginning with byte 0. If the CE is in the stop loop, the 'repeat instruction adjust' trigger is set when the REPEAT INSN switch is placed in either MPLE or SINGLE. To begin the instruction, depress START (S25).

In the repeat-single-instruction mode, the repeat-instruction microprogram is executed to set up initial conditions before entering I-Fetch of the instruction to be executed (Diagram 6-16, FEMDM). The objectives of the microprogram are to load the contents of the DATA switches into Q, set IC(21,22) to 00, inhibit updating of IC(20) or above, gate the first instruction halfword from Q to R, set STAT G, and set the 'repeat instruction initialization' latch.

When the CE is in the stop loop and the REPEAT INSN switch is in SINGLE, the 'repeat instruction adjust' trigger is set. The START pushbutton initiates the repeat-instruction function. A ROS address of 028 (hex) is forced into ROSAR to enter the repeat-instruction microprogram (Diagram 6-15). During this microprogram, STAT G is set which, in turn, sets the 'repeat instruction initialization' latch. This action prevents the loading of new instructions into Q. Note that STAT G is reset by the 'reset' micro-order that performs all the resets necessary before the next I-Fetch. The instruction that was loaded into Q from the DATA switches is executed.

Because the 'repeat instruction adjust' trigger was not reset during the initial setup routine, ROS address 028 (hex) is forced at end-op of the instruction to enter the repeat-instruction microprogram. Re-entering the repeat-instruction microprogram on each instruction resets IC(21,22) to 00, thus causing the first instruction to be repeated. Because the 'repeat instruction initialization' latch is set, Q is not loaded after the initial loading. The instruction is continuously executed until the CE is manually stopped. The TEST indicator (S48) is lit while the REPEAT INSN switch is in SINGLE.

Repeat Multiple Instructions. When the REPEAT INSN switch is placed in MPLE, the four instruction halfwords loaded into Q are continuously executed per IC(21,22). The 'repeat instruction initialization' latch inhibits data from being transferred from the SDBO to Q. Once instruction execution begins, the repeat-instruction microprogram is not entered because the 'repeat instruction adjust' trigger is reset (Diagram 6-16).

The repeat-multiple-instructions function is similar in operation to the repeat-single-instruction function except for the following:

1. The interval timer is allowed to step.
2. The 'repeat instruction adjust' trigger is reset.
3. Interruptions are executed.

Resetting the 'repeat instruction adjust' trigger prevents re-entry to the repeat-instruction microprogram. Four instruction halfwords are continuously executed per IC(21,22), until the CE is manually stopped. The TEST indicator (S48) is lit when the REPEAT INSN switch is in MPLE.

STORE PUSHBUTTON

- Allows storing data into main storage or LS from DATA switches per STORAGE SELECT switch and ADDRESS switches.

The STORE pushbutton (R32) provides a means of storing information in any address of LS or main storage. The CE must be in the stopped state (stop loop) and in state one or zero or in test, unless the CE System Interlock key is on, in order for this pushbutton to function.

The contents of the DATA switches are placed in the location specified by the ADDRESS switches and the STORAGE SELECT switch (N15) (Diagrams 6-4 and 6-10). If the STORAGE SELECT switch is in LOCAL, the five low-order ADDRESS switches (27–31) specify the LS location into which the contents (32 bits plus 4 parity bits) of the right-half of the DATA switches (32–63) are to be stored. ADDRESS switch 27, in the 0 position, permits storing into the general-purpose registers and, in the 1 (down) position, permits storing into the floating-point registers. ADDRESS switches 27 and 28, when set to 1's, address the working register.

If the STORAGE SELECT switch is in MAIN, the contents (64 data bits plus 8 parity bits) of the DATA switches are stored into main storage on a doubleword boundary per ADDRESS switches 9–31.

If the STORAGE SELECT switch is in MAIN BYTE, one byte is stored into main storage per ADDRESS switches 29–31. ADDRESS switches 9–28 specify the doubleword boundary in main storage. The value contained in the ADDRESS switches is placed into D for storing in main storage. The contents of the ADDRESS switches are placed into E (via D) for storing in LS.

For all store operations, the original contents of D, S, and T are destroyed. Correct parity is automatically generated before storing into either main storage or LS. After the data is stored, the microprogram enters the count delay routine and the CE re-enters the stop loop (Diagram 6-4).

When STORE is depressed, the pushbutton signal is ANDed with the 'STO-OB' micro-order. When the stop loop microprogram senses that STORE has been depressed, ROSAR(11) is set, causing entry into the store microprogram routine. The stop loop is re-entered after the store operation is executed.

DISPLAY PUSHBUTTON

- Allows displaying of data from main storage into ST and AB or from LS into T per STORAGE SELECT switch and ADDRESS switches.

The DISPLAY pushbutton (R35) displays the contents of any location in LS or main storage. The CE must be in stop loop and state one or zero or in test, unless the CE System Interlock key is on, in order for this pushbutton to function. The address and the storage to be used are determined by the position of the ADDRESS switches and the position of the STORAGE SELECT switch (N15), respectively (Diagrams 6-4 and 6-10). Data from main storage (64 data bits plus 8 parity bits) is displayed in ST and AB. (Set roller switches 1 and 2 to position 3 to display contents of ST, and set roller switches 3 and 4 to position 3 to display the contents of AB.) Data from LS (32 data bits plus 4 parity bits) is displayed in T. (Set roller switch 2 to position 3 to display contents of T.)

When DISPLAY is depressed, the pushbutton signal is ANDed with the 'DIS-PB' micro-order. When the stop loop microprogram senses that DISPLAY has been depressed, ROSAR(11) is set, causing entry into the display microprogram. The original contents of S, T, and D are destroyed. After the selected data has been displayed, the count delay microprogram is executed and the stop loop is re-entered.

REGISTER SELECT SWITCH

The REGISTER SELECT switch (S11) is enabled only in state zero with the TEST switch on. It is used to select the register (ATR, PSBAR, or CCR) into which data in the DATA switches will be gated by the register set microprogram (Diagram 6-4 and Figure 4-1).

REGISTER SET SWITCH

The REGISTER SET switch (Q32) provides a means by which maintenance personnel may manually store data into the ATR, PSBAR or CCR registers. It is enabled in state zero with the test switch on only. The CE must be in the Stop Loop (Manual indicator on). Depressing the REGISTER SET switch causes a branch from the Stop Loop microprogram (Diagram 6-4, FEMDM) into the Register Set microprogram. This microprogram gates the Data Switches 0–63 to the S and T registers. Data is then gated according to the REGISTER SELECT switch (Figure 4-1):

ATR: S register bits 0–31 are gated to ATR, 1, T register 32–39 bits are gated to ATR 2.

CCR: T register bits 32–63 are gated via the External register and External Bus to the CCR.

PSBAR: T register bits 41–51 are gated to Logical PSBAR bits 9–19. Logical PSBAR bits 9–12 are decoded to select an ATR slot which is then gated to Physical PSBAR. The Register Set microprogram then returns the CE to the Stop Loop.

Note: When manually loading PSBAR, data switches 32–40 and 52–63 should be set to zero. If they are not, a PSBAR parity check may result.

ROS TRANSFER PUSHBUTTON

- ROS TRANSFER pushbutton allows ROS microprogram branch to any ROS location.

The ROS TRANSFER pushbutton (Q35) allows entry into a ROS word if the CE is in state one, zero, or test, regardless of the CE System Interlock key position. Depressing ROS TRANSFER places the contents of the 12 high-order ADDRESS switches into ROSAR (Diagram 6-4 and 6-17, FEMDM). The next microinstruction is taken from ROS and placed in the ROS sense latches. Further action now depends upon the position of the RATE switch.

If the RATE switch (Q25) is in PROCESS, the CE continues executing ROS words from the entry point. If the RATE switch is in INSN STEP, the CE continues until an end-op is reached.

If the RATE switch is in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT, the CE stops with the ROS word specified by the ADDRESS switches contained in the sense latches. (This ROS word may be displayed by means of the appropriate indicators.) Depressing START (S25) advances ROS one cycle, and the contents of the ROS data register may then be displayed. If START is depressed again, ROS advances as in the single-cycle mode.

Regardless of the position of the RATE switch, checks may occur as a result of storage data bus transfer to registers and from the registers through the parallel adder. To prevent the CE from stopping on these checks, place the CHECK CONTROL switch (N17) in DSBL. (See "CE CHECK CONTROL SWITCH".)

When ROS TRANSFER is depressed, the pushbutton signal is ANDed with the 'ROS-PB' micro-order. When the stop loop microprogram senses that ROS TRANSFER has

been depressed, ROSAR(11) is set, causing entry into the ROS transfer microprogram. Instruction execution continues from the ROS address entered into the 12 high-order ADDRESS switches.

STORAGE-RIPPLE MICROPROGRAM

- The storage-ripple microprogram allows continuous storing of data into, or displaying of data from, all locations of either main or local storage.

The storage-ripple microprogram (Diagram 6-18, FEMDM) is capable of (1) storing data from the DATA switches in all addresses in LS or main storage and placing correct parity in the storage-protect keys or (2) reading all locations of LS or main storage and displaying the data. Main storage or LS is chosen by means of the STORAGE SELECT switch (N13).

If main storage is selected, the storage-ripple microprogram begins at address 0 and continues until an invalid address is detected, at which point a restart beginning at address 0 occurs. Diagram 6-18 shows the restart of an interruption request that resulted from detecting an invalid address. If LS is selected, the storage-ripple microprogram begins at address 0 and loops through all addresses in LS. Manual intervention (e.g., system reset or IPL) is required to exit from the storage-ripple microprogram.

The storage-ripple microprogram may also be used in troubleshooting by loading all storage locations with a predetermined value and then reading back the data.

Storage-Ripple-Store Function

This function allows storing data in all locations of LS or main storage. To accomplish this, the CE must be in the stop loop. Enter the data to be stored into the DATA switches, and position the STORAGE SELECT switch (N15) to select LS or main storage. Enter 800006 (hex) into ADDRESS switches 0–23, and depress ROS TRANSFER. The data previously entered into the DATA switches is continually stored in all locations in main or local storage. Also, correct parity is placed in the storage protect keys on main storage ripple. Incorrect data may be stored if the DATA switches are changed when ROS is in the storage-ripple-store routine. To terminate the routine without causing bad data in storage, momentarily place the STORAGE SELECT switch in LOCAL; then depress SYSTEM RESET (P30). The ROS microprogram for the storage-ripple-store function is shown in Diagram 6-18.

Because there is no automatic means of clearing main storage or LS, the storage-ripple-store microprogram may be used for this purpose.

Storage-Ripple-Display Function

This function allows reading and displaying of data from all locations in LS or main storage. The storage-ripple-display microprogram continuously reads all locations in LS or main storage as determined by the setting of the STORAGE SELECT switch (N13). The CE must be in the stop loop before ROS TRANSFER is depressed. To execute the storage-ripple-display routine, enter 800000 (hex) into the ADDRESS switches, select main storage or LS, and depress ROS TRANSFER (Diagram 6-18). If LS is selected by means of the STORAGE SELECT switch, the data is contained in S and PAL (32–63). If main storage is selected, the data is contained in AB and ST. The data may then be displayed, using roller switches. Data is checked in PAL for correct parity.

To terminate the routine, depress SYSTEM RESET (P30).

STOP ON ROS ADDRESS/REPEAT ROS ADDRESS SWITCH

The STOP ON ROS ADDRESS/REPEAT ROS ADDRESS switch (Diagram 6-17) is active only in state one or zero or in test mode. The test indicator is lit when this switch is in either active position.

The STOP ON ROS ADDRESS position causes the CE to stop when it gets a successful compare between the ROS Address register and Address keys 8–19. The selected ROS address is displayed in the previous ROSAR register, roller 1 position 4. The ROS word bits are displayed in the ROS Data register, position 4 of rollers 2, 3, and 4. When a ROS address compare stop has stopped the CE, single-cycle mode may be entered. Maintenance personnel should be aware that stopping the CE in a storage cycle may cause loss of data.

The REPEAT ROS ADDRESS position causes the continuous reading out of a ROS word (Diagrams 6-17 and 6-4). The ROS word address is specified by Address keys 8–19. When the desired ROS address is entered in the Address keys and ROS TRANSFER is depressed, continuous machine cycles are taken, fetching the specified ROS word on each cycle. All storage requests are blocked. Changing the selected ROS address (Address keys 8–19) while the CE is cycling in the REPEAT ROS ADDRESS function may cause ROS parity checks.

SYSTEM AND SUBSYSTEM PSW RESTART AND WAIT STATE

The performance of system or subsystem PSW restart functions result from depression of the PSW RESTART

pushbutton on the CE control panel. They occur with the System Interlock key on or off, respectively, and perform a system or subsystem reset accordingly. The PSW RESTART pushbutton is active only in state one or zero or in test, unless the CE System Interlock key is on. Each forces the CE into the stop loop and causes the IPL – PSW microprogram branch from the stop loop (Diagrams 6-4 and 6-8, FEMDM). In addition, a system PSW restart causes the CE to configure elements of a subsystem (per MAIN STORE and LOAD UNIT SELECT switches) to itself and to each other in state three. Any other elements not in test are configured to the selected subsystem in state three. The PSW from PSA location 0 is then loaded (Diagram 5-601, FEMDM).

After the new PSW is fetched, the CE continues processing if the RATE switch (Q25) is in PROCESS.

The 'stop' and 'manual' triggers are reset at the beginning of the PSW-restart microprogram (Diagrams 6-4 and 6-8). The PSW RESTART pushbutton causes entry to the normal load PSW routine, which refills Q, R, and E.

At every end-op, PSW(14) is tested (Diagram 6-19, FEMDM). If PSW(14) = 1, the Wait state is entered [If PSW(14) = 0, the CE is placed in the Running state]. The wait-state routine is depicted in Diagram 6-20, FEMDM. If the interval timer is to be stepped, the wait microprogram loop is re-entered after stepping the timer. If STOP (S30) is depressed, the stop loop is entered. When a restart from the stop loop is executed, the Wait state is re-entered if PSW(14) = 1.

If an interruption causes a new PSW to be loaded into the CE, PSW(14) is again tested, and the Wait state is re-entered if PSW(14) = 1; otherwise, the CE is placed in the Running state.

When PSW RESTART is depressed, the pushbutton signal is ANDed with the 'IPL-PSW' micro-order. When the stop loop microprogram senses that PSW RESTART has been depressed, ROSAR(11) is set, causing entry to the PSW restart microprogram. A new PSW is loaded, and program execution continues.

DISABLE INTERVAL TIMER SWITCH

The DISABLE INTERVAL TIMER switch (N11), when placed in the down position, prevents the interval timer [main storage PSA location 80, decimal (50 hex)] from being advanced (Diagram 6-21, FEMDM) only if the CE is in state zero or in test, regardless of the CE System Interlock key position. In the center position, the timer is stepped at regular predetermined intervals.

In addition to the switch, the timer is disabled when the Diagnose instruction sets MCW(20) (disable timer bit) or when operating in the:

1. Stop-loop routine.
2. Single-cycle mode.
3. Instruction-step mode.
4. Repeat single-instruction mode.
5. Scan mode.

The DISABLE INTERVAL TIMER switch is inactive when the PULSE MODE switch (N21) is in the TIME position and the 'pulse mode initialization' trigger is set. When this switch is in the down position, the TEST indicator (S48) is lit.

Note: Do not disable the interval timer when operating in multiprogramming or multiprocessing mode.

INTERRUPT PUSHBUTTON

The INTERRUPT pushbutton initiates an external interruption by setting the 'console signal' trigger. If PSW(7) = 1, an interruption is taken after the current instruction, and interruptions of higher priority are completed. If PSW(7) = 0, the interruption request remains pending.

During the interruption, PSW(25) is made a 1, indicating that the INTERRUPT pushbutton is the source of the interruption. This pushbutton is effective only if the CE is in state one or zero or in test, unless the CE System Interlock key is on.

CE CHECK CONTROL SWITCH

The CE CHECK CONTROL switch (N17), a three-position toggle switch, controls the system when a machine check is encountered. The machine checks that set the error trigger and the logic controlled by the CE CHECK CONTROL switch are shown in Diagram 6-22, FEMDM. The TEST indicator (S48) is lit when this switch is in any position other than PROC.

When the CE CHECK CONTROL switch is in PROC and a machine check is detected, and PSW(13) = 1 (machine check mask bit), the CE is stopped and the machine status is logged out to main storage. A machine check interruption is then executed. If PSW(13) = 0, the check is ignored and no logout or interruption occurs.

When CE CHECK CONTROL is in STOP and a machine check is detected, the CE clock is stopped and no logout occurs. The error trigger is set; the type of error may be determined by examining the roller switches. If CHECK RESET (P35) is depressed, operation is resumed, but the results may be unpredictable.

When CE CHECK CONTROL is in DSBL and a machine check is detected, logout and interruptions do not occur, and the operation is not terminated. Program execution continues, ignoring machine checks.

INHIBIT CE HARDSTOP SWITCH

The INHIBIT CE HARDSTOP switch is active only in state zero or in test. This switch prevents a CE hardstop which would otherwise result from errors which cause the CE to send ELC to other CEs in the system. Operation of this switch does not interfere with the normal handling of most CK REG 1 or 2 monitored errors per the CE CHECK CONTROL switch.

PULSE MODE SWITCH

The PULSE MODE switch (N20) provides a means of looping through a number of machine cycles, starting at a selected address, or of looping each time the interval timer is advanced. This switch has three positions: PROC (process), normal CE operation; TIME; and COUNT.

The CE must be in the stop loop before entering or leaving pulse mode operation. The PULSE MODE switch is inoperative during repeat-instruction mode.

Two triggers control pulse mode operation: 'pulse mode adjust' trigger and 'pulse mode initialization' trigger (Diagram 6-23, FEMDM). The 'pulse mode adjust' trigger determines when to force an overriding branch to the pulse-mode-initialization microprogram and when to reset the CE. The 'pulse mode initialization' trigger is set by depressing START (S25) with the 'pulse mode adjust' trigger set. As a result, pulse mode operation begins.

The TEST indicator (S48) is lit when the PULSE MODE switch is in any position other than PROC.

PROC Position

The PROC position is used during normal program execution.

TIME Position

- Load program into main storage.
- Place starting address of program into main storage bytes 5–7 of PSA location 0 by means of ADDRESS switches.
- Enter stop loop.
- Place PULSE MODE in TIME position.
- Depress START.

When the PULSE MODE switch is in the TIME position, instruction execution begins at the address specified in the address portion of the doubleword located in PSA location 0 of main storage. Therefore, the starting address must be loaded (by means of the ADDRESS switches) into bits 40–63 of the doubleword located at PSA location 0 before depressing START (S25). Entering data manually into main storage (from the DATA switches) must be done with the CE in the stop loop and with STORE (R32) depressed. The RATE switch (Q25) must be in the PROCESS position.

The initial setup conditions are:

1. The program is in main storage.
2. The starting address is in main storage bytes 5–7 of PSA location 0.
3. The CE is in the stop loop.
4. The PULSE MODE switch is in the TIME position.

A flowchart of the pulse mode operation is shown in Diagram 6-24, FEMDM. When the CE is in the stop loop and the PULSE MODE switch is set to TIME, the 'pulse mode adjust' trigger is set (Diagram 6-23). Depressing START sets the 'pulse mode setup' latch. This action fires a 350-ns singleshot that initiates the pulse-mode function. The 'pulse mode initialization' and the 'force address' triggers are set. This action, coupled with the set 'pulse mode adjust' trigger, sets ROSAR(9,11) to force an address of 005 into ROSAR. A Machine reset is generated. Note that the 'reset manual control' signal is inhibited (Diagram 6-23). The CE then enters the pulse-mode microprogram. The 'pulse mode adjust' trigger is reset by the pulse-mode microprogram. The loading of the count into the MCW is meaningless. Instructions are then executed until the interval timer is stepped, at which time the 'pulse mode adjust' trigger is set. Because the 'pulse mode initialization' trigger is set, a system reset occurs, an address of 005 is forced into ROSAR, and the pulse-mode microprogram is re-entered. This action results in executing the program from clock step to clock step. Looping through the pulse-mode microprogram and the main storage program continues until manually stopped.

COUNT Position

- Load program into main storage.
- Place starting address of program into main storage PSA 0 bytes 5–7 by means of ADDRESS switches.
- Enter stop loop.
- Place PULSE MODE in COUNT position.
- Enter number of machine cycles [up to 7FF (hex)] to be executed into DATA switches 53–63.
- Depress START.

When the PULSE MODE switch is in the COUNT position, instruction execution begins at the address specified in the address portion of the doubleword located in PSA location 0 of main storage, and proceeds through the number of machine cycles entered into DATA switches 53–63. Each time the cycle counter is reduced to 0, a machine reset occurs and the program is re-entered.

Except for PULSE MODE being in the COUNT position and the count entered into DATA switches 53–63, the initial setup conditions are similar to the time-mode routine. As shown in Diagram 6-24, the same microprogram is executed for both COUNT and TIME positions. The program in main storage is entered at end-op of the microprogram. The cycle counter is reduced by 1 on each machine cycle. When the counter equals 0, a machine reset occurs, and the pulse-mode microprogram is again executed. The looping through the microprogram and the storage program continues until manually stopped.

360 MODE SWITCH

- 360 mode allows operation of System 360 programs on a 9020D or E subsystem.

- 360 mode operation can use only IOCE 1 for I/O operations.

When running System/360 programs on a 9020D/E subsystem (comprising a CE, one or more SEs, and IOCE 1), it is necessary to inhibit certain actions which are not compatible with System/360 operating programs.

The 360 mode indicating switch on the CE operator's panel turns the '360 mode' latch on if the CE is in state one, zero, or in test. It accomplishes the following when on:

1. Causes the subsystem to operate as through bit 6 of the CCR (Inhibit Logout Stop) is set on. The system cannot perform split logout or issue logout stop to an SE. PSBARs are set to zero and cannot be stepped. (PSA is not relocatable.)
2. Inhibits manipulation of PSW bits 16–19 (extended system mask) on Load PSW and Set System Mask.
3. Causes a PSA lockout to create a machine-stop condition.
4. Causes an SE-stopped condition to create a machine-stop condition.
5. Causes all unique 9020D/E System instructions to be invalid.
6. Causes I/O processor operations to cease.

Only IOCE 1 will operate in 360 mode and only if the CE and IOCE 1 are configured to each other. If IOCE 1 is not configured to the CE in 360 mode, the CE is unable to execute I/O operations.

When in 360 mode, depressing the 360 Mode switch returns the subsystem to 9020 mode. 360 mode is also released by the following:

1. An External Start issued to the CE.
2. A CE Power-On reset.
3. A reconfiguration which places the CE in state two or three.
4. FLT Load.

LOG OUT PUSHBUTTON

The LOG OUT pushbutton (S35) logs the machine status into fixed locations of the PSA. This pushbutton is active only in state one, zero, or in test unless the CE System Interlock key is on. It is enabled by the 'manual' trigger (stop loop) or 'not pass' pulse (clock stopped). The logic associated with the LOG OUT pushbutton is shown in Diagram 6-25, FEMDM. Refer to Section 2 of this chapter for a complete discussion of logout.

SCAN MODE, ROS/PROC/FLT SWITCH

The SCAN MODE, ROS/PROC/FLT switch (N5) takes the CE out of program control for FLTs or ROS tests. It has three positions:

1. ROS: When in this position and LOAD (S51) is depressed, ROS tests are read into the selected SE by the IOCE selected by the LOAD UNIT switches. Signals from the IOCE cause the tests to be executed in the CE.
2. PROC – normal position: Normal CE processing takes place.
3. FLT: When in this position and LOAD is depressed, FLTs are read into the selected SE from the IOCE selected by the LOAD UNIT switches. Signals from the IOCE cause the tests to be executed in the CE.

Logic associated with this switch is shown in Diagram 6-26, FEMDM. The interval timer is disabled and the TEST indicator (S48) is lit when this switch is in the ROS or FLT position. Refer to Section 2 of this chapter for a complete discussion of FLT/ROS tests.

SCAN MODE, REPEAT SWITCH

The SCAN MODE, REPEAT switch (N1) is used to continuously repeat the ROS test or FLT in main storage or the new test being sought. When this switch is in the REPEAT (down) position, depressing START (S25) causes the ROS test or FLT in main storage to be executed repeatedly by forcing the ROS test sequencer to 3.

BACKSPACE FLT PUSHBUTTON

The BACKSPACE FLT pushbutton (S33) is gated by ROS or FLT Test mode. It allows backspacing of the test tape by one record with each depression. The pushbutton sets the 'FLT backspace' trigger which resets the CE and sends subsystem reset to the IOCE and the line FLT BACKSPACE IOCE. See Diagram 6-27, FEMDM. Testing may be restarted by performing a ROS test or FLT IPL.

FREQUENCY ALTERATION SWITCH

The FREQUENCY ALTERATION switch (S5) is used to increase the CE clock frequency. When this switch is in the DISABLE position, each machine cycle is 200 ns (normal speed). With the CE in state zero and test and the FREQUENCY ALTERATION switch in the ENABLE position, the CE clock cycle is decreased to 195 ns ±1/2 ns (Diagram 4-3, FEMDM).

LAMP TEST/ALLOW INDICATE PUSHBUTTON

The lamp test function of the LAMP TEST/ALLOW INDICATE pushbutton is enabled any time the CE is powered on. This function provides a lamp test for those indicators on the CE control panel which are not associated with the roller switches or power. These include the four state indicators, 360 MODE, CK REG 1 SUMMARY, CK REG 2 SUMMARY, SYSTEM, MANUAL, WAIT, TEST, and LOAD indicators. Duplicate lamps on the SC or CC are also tested by the LAMP TEST function: state indicators, MANUAL, WAIT, LOAD, and LOGIC CHECK (CK REG 1 or 2).

The allow-indicate function of the LAMP TEST/ALLOW INDICATE pushbutton is enabled only in state one or zero or in test mode. When the CE hardstops during a logout (i.e., because of a storage error), all the roller indicators are on. The allow-indicate function allows the roller indicators to display the state of the machine instead of having all indicators on. This function is also useful when single-cycling through scan-in or logout.

Allow-indicate prevents the CE logic from using the roller indicator buses as a data path; therefore, use of this function while scan-in or logout data is being gated from the indicator buses will invalidate the data. It should not be used unless the CE is either hardstopped or in single-cycle mode.

INDICATE RLR 1 POSITION 6

This switch (N27) selects either the L-register or the M-register for display if position 6 of roller 1 is selected.

7201-02 CE CONTROL PANEL INDICATORS

On panel E of the system control panel, there are six rows of indicators, with 36 indicators (implicitly numbered 0–35) in each row (Diagram 6-1). Associated with each row of indicators is a six-position roller switch. The operator may display the contents of a register or the status of a trigger or latch by placing the proper roller switch in the correct position. A roll chart above each row of indicators shows the information being displayed for each indicator. As the roller position is changed, the roll chart rotates to correspond with the roller position. The roller switch, the positions and the information displayed for each row of indicators are shown in Diagram 6-2.

A lamp test for the roller switches is provided by positioning these switches between detent positions.

Other indicators on the CE control panel indicate machine status, check conditions, and power status, as follows:

1. SYSTEM (S45) – Lights when the CE is neither in the wait state, stop loop, nor has the 'pass pulse' trigger reset.
2. MANUAL (S46) – Lights when the CE is in the stopped state. The CE is executing the stop loop microprogram.
3. WAIT (S47) – Lights when the CE is in the wait state. The CE is stopped in the wait microprogram if the MANUAL indicator is not lit. If this indicator is lit, the 'stop' trigger has been set and the CE is in the stop loop.
4. TEST (S48) – Lights when a manual control is not in a position for normal processing or a maintenance function is being applied, as follows:
   a. The RATE switch (Q25) is in a position other than PROCESS.
   b. The CHECK CONTROL switch (N19) is in STOP or DSBL.
   c. The DISABLE INTERVAL TIMER switch (N13) is down.
   d. The ADDRESS COMPARE switch (N15) is down.
   e. The PULSE MODE switch (N21) is in TIME or COUNT.
   f. The SCAN MODE, ROS/PROC/FLT switch (N3) is in ROS or FLT.
   g. The REPEAT INSN switch (N23) is in MPLE or SINGLE.
   h. The DEFEAT INTERLEAVING switch (N9) is in NO REV or REV.
   i. ROS ADDRESS switch (N25) is set to any position other than PROC.
   j. The Diagnose instruction is active.
   k. The LAMP TEST pushbutton is depressed.
5. LOAD (S49) – Lights when the CE is in a load state (IPL microprogram). The LOAD indicator is turned off after a successful load.

6. CHECK REG 1 (K52) – Lights on any CHECK REG 1 error. The roller switches must be examined to determine the specific error.
7. CHECK REG 2 (M52) – Lights on any CHECK REG 2 or HARD STOP error. The roller switches must be examined to determine the specific error.
8. STATE THREE, TWO, ONE, ZERO (H2–6) – Indicate the CE state as decoded from the CE CCR bits 0 and 1. State is set by a SCON instruction, System IPL (state 3) or System PSW Restart (state 3).

POWER CONTROLS AND INDICATORS

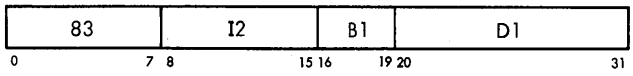
For CE power indicator and control information, refer to the 9020 D/E Power Controls and Distribution FETOM.

The 7201-02 maintenance facilities available to maintenance personnel are: (1) Diagnose instruction and associated MCWs; (2) Resident micro-diagnostic; (3) Logout, ROS tests, and FLTs; (4) ripple tests; (5) diagnostic programs; and (6) marginal checking. For a discussion of the CE control panel, refer to Section 1 of this chapter.

DIAGNOSE INSTRUCTION AND MCWS

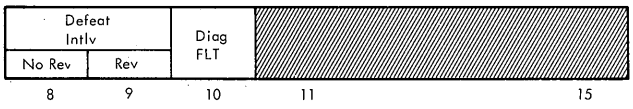
The Diagnose instruction (Section 8 of Chapter 3) has two purposes: (1) It is available to the diagnostic programmer as a maintenance aid when the CE is in state zero. (2) It is available to the system programmer for 9020 operations when the CE is in state three, two, or one.

The Diagnose instruction has an SI format with an op code of 83:



The operations that the Diagnose instruction can perform (when used by the diagnostic programmer with the CE in state zero) are selected by the I2 field of the instruction and by the bit configuration of the doubleword addressed by the storage operand address (contents of GPR addressed by B1, + D1). This doubleword is the MCW and bears the same relation to the Diagnose instruction as the CCW does to an I/O instruction. That is, the Diagnose instruction addresses the location of the MCW, and the MCW specifies the machine function. Note that an exception to this analogy is the I2 field, which can also designate certain diagnostic functions.

The bits of the I2 field have the following meaning:



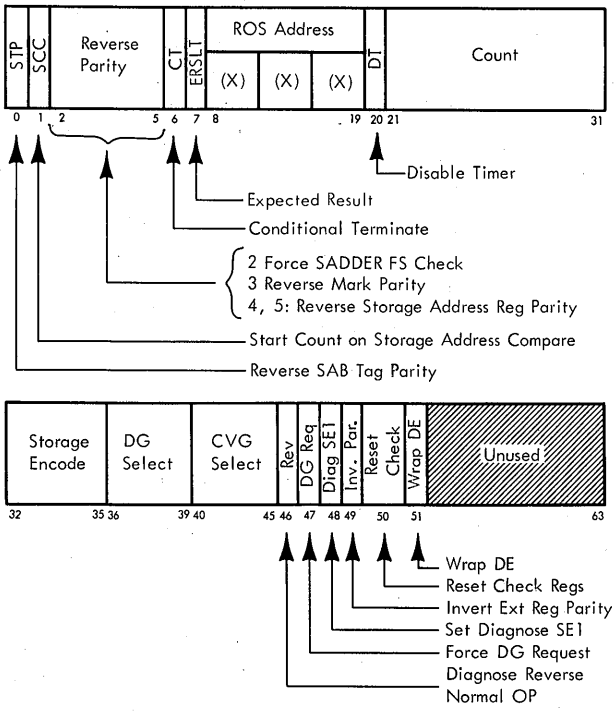
1. Bit 8, Defeat Interleaving and No Reversal of Storage Addresses. Interleaving is disabled, and no even/odd storage area reversal occurs.
2. Bit 9, Defeat Interleaving and Reverse Storage Addresses. Interleaving is disabled, and the even and odd storage areas are reversed.
3. Bit 10, Diagnose FLT. Allows portions of FLTs to be executed under control of the Diagnose instruction. While the FLTs are being executed, special CE functions are generated, and storage requests and clock are inhibited. This bit, which is used in conjunction with the setting of address 6B0 in ROSAR, simulates the FLT position of the SCAN MODE, ROS/PROC/FLT switch on the CE control panel.
4. Bits 11–15. Spares.

THE MCW

When the Diagnose instruction is executed, the MCW used is a 64-bit word in the main storage location specified by the storage operand address. It designates the diagnostic function(s) to be performed. The MCW is a 32-bit word when used to control scan operations. For ROS tests and FLTs (Scan operations), the MCW is contained in word 1 of each test on the ROS or FLT test tape and specifies the control conditions necessary for the test, such as the expected result and the ROS word to be used for gating control. A different format is used for each MCW.

Diagnose Instruction MCW for CE in State 0

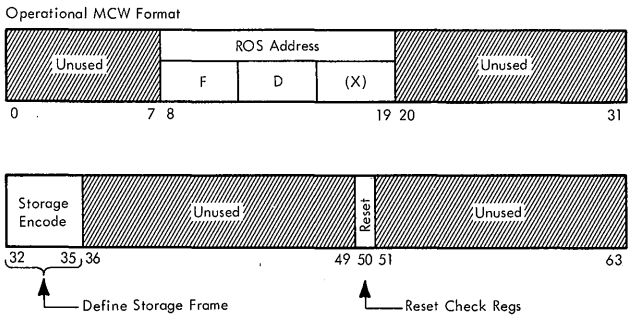
If executing the Diagnose instruction when the CE is in state zero, the MCW applies as follows:



1. Bit 0, Reverse SAB Tag Parity.
2. Bit 1, Start Count on Storage Address Compare. When used with bit 6, the FLT counter does not begin decrementing until the SCI addresses the same location as set into the MAIN STORE ADDRESS COMPARE (ADDRESS switches 9–28) switches.
3. Bit 2, Reverse Serial Adder Full-Sum Parity. Reverses the parity bit in the full-sum latch of the serial adder, thus allowing a test of the parity-checking circuits.
4. Bit 3, Reverse Mark Parity. Reverses the parity of the mark bits being sent from the SCI to main storage, thus allowing a test of the mark parity-checking circuits in main storage.
5. Bits 4 and 5, Reverse SAR Parity. Cause the parity bits which are sent to the storage address register to be reversed as follows:
   00 – No parity reversal.
   01 – Reverse low-order parity bit.
   10 – Reverse next-higher parity bit.
   11 – Reverse high-order parity bit.
6. Bit 6, Conditional terminate (Log on count). Causes a logout to main storage when the FLT counter reaches 0. At the conclusion of the logging operation, the CE performs a machine check interruption.
7. Bit 7, Expected result (ERSLT).
8. Bits 8–19, ROS Address. When the Diagnose instruction has completed its execution phase, these address bits are placed in ROSAR, and the operation branches to this location. The address placed in ROSAR can specify any location in ROS. Refer to Diagram 5-609, Sheet 3, FEMDM, for the most frequently used ROS address.
9. Bit 20. Disable interval timer.
10. Bits 21–31, Count. Specify the number of cycles that are to occur before the CE enters a logout routine. The count field is as follows: MCW(21–25) is loaded into the address sequencer; MCW(26–29) is loaded into the FLT counter; MCW(30, 31) is loaded into the FLT clock. By combining the address sequencer, the FLT counter, and the FLT clock, a maximum count of 2047 (11 bits) can be obtained. (These three counters are combined only when the MCW is used with the Diagnose instruction.) The FLT clock controls the decrementing of the FLT counter and address sequencer by 1; when the three counters combined equal 0, the logout routine is started.
11. Bits 32–35. Used to select a DE during 'Force DG Request' and 'Wrap DE'.
12. Bits 36–39. Used to select a DG during 'Force DG Request' and 'Wrap DE'.
13. Bits 40–45. Used to select a CVG during 'Force DG Request' and 'Wrap DE'.
14. Bit 46. Reverse normal op.
15. Bit 47. Used to 'Force DG Request' (with bits 32–45).
16. Bit 48, Set diagnose SE 1. Blocks generating 'invalid address' decoded, when SAB bit 0=1.
17. Bit 49, Invert parity. Used to reverse external register parity for byte 0 and SDBI parity for byte 4, to allow gating bad parity into the receiving element's CCR using a SCON instruction to check the receiving element's ability to recognize bad CCR parity.
18. Bit 50, Used to reset any bits on in the Check registers.
19. Bit 51, Used with bits 32–45 to initiate 'Wrap DE'.

Diagnose Instruction MCW for CE in State Three, Two, or One

When the Diagnose instruction is executed with the CE in state three, two, or one, the MCW applies as follows:

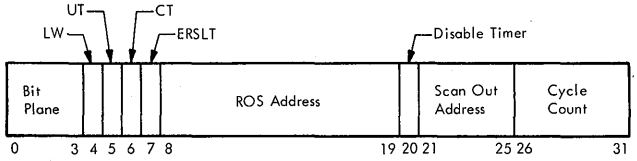


1. Bits 0–7. Not used.
2. Bits 8–19, ROS address. (Must contain FD in bits 8–15 or a program specification error ensues.) Refer to Diagram 5-609, FEMDM.
3. Bits 20–31. Not used.
4. Bits 32–35. Used to select a storage frame when executing the logout storage operational kernel.
5. Bits 36–49. Not used.
6. Bit 50. Used to reset any bits on in the Check registers.
7. Bits 51–63. Not used.

Note: The I2 field on the Diagnose instruction is also degated and has no effect when the CE is in state three, two, or one.

ROS Test MCW

The MCW for ROS tests, the right half of word 1 of each test, contains the following control information about the test:

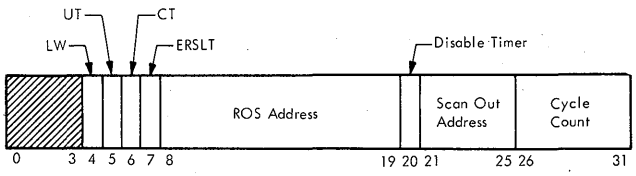


1. Bits 0-3, Bit Plane. Contain the number of the ROS bit plane tested. These bits are for display only; the ROS word is selected by the ROS address in bits 8–19, and the bit to be tested is selected by the mask.
2. Bit 4, always off, since the right word always contains the trigger status for ROS tests.
3. Bit 5, Unconditional Terminate (UT). If this bit equals 1, the test always causes a stop.
4. Bit 6, Conditional Terminate (CT). If this bit equals 1 and an error is encountered, testing is terminated. However, if it equals 1 and the test does not detect an error, the CE continues with the next ROS test. If this bit equals 0, termination is dependent upon the status of MCW(5).
5. Bit 7, Expected Result (ERSLT). Defines what the status of the ROS bit being tested should be. Comparing the ROS bit with the 1 or 0 in this bit determines whether the test passed or failed.
6. Bits 8–19, ROS Address. Contain the ROS address of the micro-instruction that contains the bit to be tested.
7. Bit 20, Inhibit Timer. Used for display only; indicates a successful stop in hardcore.
8. Bits 21–25, Scan Out Address. Specify the scan-out address (roller switch position) of the portion of the ROSDR that contains the bit to be tested. This address will be either 15, 16, 17, or 19.
9. Bits 26–31, Cycle Count. Determine the number of CE clock cycles ROS must cycle after scan-in and before scan-out.

FLT MCW

The MCW in the right half of word 1 of every FLT contains control data about the test being performed. Pertinent bits of the MCW are retained in the MCW register during the running of an FLT. Other bits set the address sequencer, determine the ROS starting address, and fix the number of clock cycles the CE will take following scan-in.

Following is the format of an MCW during an FLT:



1. Bits 0–3. Not used.
2. Bit 4, LW. Defines whether the left half or right half of the doubleword to be scanned out contains the 'exit' trigger status. If bit 4 equals 1, the left half is the desired word. If bit 4 equals 0, the right half contains the trigger status.
3. Bit 5, UT. Always causes a stop when it equals 1.
4. Bit 6, CT. May cause a stop, depending on the outcome of the test. If this bit equals 1 and an error is encountered, testing is terminated. This bit is always set in zero-cycle and one-cycle FLTs currently in use.
5. Bit 7, ERSLT. Defines what the status of the 'exit' trigger should be. Comparing the 'exit' trigger state with the 1 or 0 in this bit determines whether the test passed or failed.
6. Bits 8–19, ROS Address. Contain the ROS address of the ROS word to be used for gating control.
7. Bit 20, Inhibit Timer. Used for display only; indicates a successful stop in hardcore.
8. Bits 21–25, Scan Out Address. Specify the scan-out address of the scan-out word containing the status of the 'exit' trigger.
9. Bits 26–31, Cycle Count. Fix the number of times the CE is to cycle following scan-in.

RESIDENT MICRO-DIAGNOSTIC

The resident micro-diagnostic is a ROS-controlled sequence for checking CE registers and data paths. The program may be started (providing the CE is in state zero or one and in manual mode) by doing a ROS transfer to address FAA. The program will loop on itself, alternately using an all 0's and an all 1's pattern to transfer from one register to another. Set Address key 29 on to cause the program to load data from Data keys (0–63) instead of using all 0's, all 1's patterns.

Set CHECK CONTROL switch to STOP position to stop on an error (active only in state zero). Failures show up as adder checks, local store bus checks, SDBI checks, E-register parity errors, ROS parity errors, or multiply decode parity errors. The following registers and functions are tested: S, T, A, B, Local Store, K, D, IC, F, G, Select, External, DAR Mask, M, N, Q, R, E, and multiple decode.

If parity errors occur in Q, R, N, or E registers, set Address key 30 on to cause the program to load good parity. MAIN STORAGE SELECT switch must address a configured SE. Refer to LADS AA001.

INTRODUCTION TO LOGOUT, ROS TESTS, and FLTS

Logout stores the status of the CE control panel indicators in fixed positions of main storage when a trouble symptom occurs; the data logged out may be subsequently recalled for analysis, although the status of the indicated logic is changed from what it was when the symptom appeared. ROS tests check each bit position of every ROS word. FLTs check the CE at the logic block level.

Logout, ROS tests, and FLTs are implemented by special hardware called scan logic. Therefore, when employing these maintenance aids, the CE is said to be in the scan mode.

The scan logic performs the following:

1. Controls the operation of FLTs.
2. Records the state of the CE when a machine malfunction is detected (logout).
3. Executes the Diagnose instruction.
4. Controls the operation of the ROS tests.

In the discussion that follows, certain terms are used which refer only to scan operations. These terms are defined as follows:

1. Scan mode. The CE is said to be in "scan mode" during those operations that use the scan logic; i.e., FLT, ROS tests, and logout.
2. Scan in. The process of loading CE register positions and control triggers with a predetermined bit pattern from storage.
3. Scan out. The sampling of the status of certain triggers via a special data path from the indicator logic to PAL.
4. Logout. The function that transfers status indicators and register contents to storage when signaled by the CE.

SCAN LOGIC FUNCTIONAL UNITS

The scan logic functional units (Diagram 6-101, FEMDM), in conjunction with the CE operational hardware and the ROS microprogram, control the CE during FLT, ROS tests, logout, and Diagnose instruction execution. During these operations, the scan logic supplements and sometimes overrides the operational CE logic. In addition, the scan logic provides direct data flow paths (scan in) into triggers not ordinarily having direct entry paths and from indictable storage devices to PAL (scan out).

During scan operations, CE timing is controlled by two clocks (scan clock and FLT clock), and sequential operation is determined by three counters: address sequencer, FLT counter, and ROS test sequencer. These counters time-share the scan counter latches and decrementer. That is, although each counter is used for a specific function at a specific time, its contents are all decremented by the scan counter decrementer. For certain operations, the address sequencer and the FLT counter can be logically joined to triggers in the FLT clock to form an 11-bit cycle counter.

The five-bit address sequencer contains the low-order bits of storage word addresses, and its decoded value controls the gating signals to the scan-in or scan-out logic. The address sequencer also forms the high-order five bits of an 11-bit counter (as mentioned previously) for the Diagnose instruction and pulse-mode functions. The six-bit FLT counter counts CE clock signals when required; the low-order two bits of the FLT counter, which are contained in the FLT clock, count oscillator signals reaching FLT controls. The three-bit ROS test sequencer sequences the scan hardware during ROS tests.

Each ROS test or FLT has an associated MCW which allows the test to be controlled by a test pattern on tape. MCWs, including the Diagnose instruction MCW, are stored in the scan functional units and decoded by scan hardware.

Scan Timing

Scan timing is controlled by two clocks, the scan clock and the FLT clock. The scan clock is synchronized with the CE clock and produces clock and not-clock signals similar to the CE clock. Normally, the scan clock runs continuously, whether the CE is in scan mode or functional operation, and can be made to run with the CE clock off. The output of the scan clock steps the FLT clock.

The FLT clock is used in conjunction with the scan clock to provide scan control timing. It is synchronized with the scan and CE clocks and provides four output signals, FLT-time 0–3. The two FLT clock triggers can be logically combined with the address sequencer and the FLT counter to form an 11-bit cycle counter.

Scan Clock Highlights

- Consists of inverters which delay 'gated oscillator D' signal from CE clock, and therefore is a symmetrical clock.
- Distributes clock and not-clock signals with delay levels of P0–3 to P2.
- Runs continuously unless blocked by SCI.
- During scan operations, CE clock is turned off, leaving scan clock in control.

The scan clock (Diagram 6-102, FEMDM) consists of a number of inverters, each of which delays a 'gated oscillator D' signal from the CE clock approximately 10 ns. The output of each delaying inverter is distributed as a clock or not-clock signal with a delay level of from P0–3 to P2. (See Chapter 2, Section 1, for an explanation of clock, not-clock, and delay levels.) The scan clock runs continuously unless stepped by one of the following actions:

1. The 'pass pulse' trigger is reset (which also stops the CE clock). This trigger is set when any operation is initiated from the CE control panel and normally remains set throughout most CE operations. However, it can be reset

by one of the following conditions if the SCI is not holding the CE clock on:

a. The UT bit is set.
b. The CT bit was on and the 'fail' trigger was set during a test.

Note: The above two conditions reset the 'pass pulse' trigger at FLT-clock-2 time if the ROS test sequencer is at 0.

c. The 'block' trigger is set. This trigger is set if one of the pushbuttons listed in Diagram 6-102 is depressed in single-cycle mode.

2. The 'stop clock' trigger is set with the RATE switch not in the SINGLE CYCLE STORAGE INHIBIT position and with the TEST MODE, REPEAT switch not set. The 'stop clock' trigger is set by the 'CPU 2' latch or the 'insert key' trigger.
3. The 'STOP 1' or 'STOP 2' micro-orders are executed.

Because the scan clock is driven by a 'gated oscillator D' signal (basic clock signal delayed; Diagram 4-1, FEMDM), it is always in synchronization with the CE clock. The scan clock signals, P0–3, P0–2, P0–1, P1, and P2 (Diagram 6-102), are developed by passing the 'gated oscillator D' signal through inverters. Each inverter delays the signal approximately 10 ns. The P0–3 signal is the first scan clock output and is developed after being delayed by an adjustable time delay. (The time delay should be adjusted so that P0 from the scan clock coincides with P0 from the CE clock; see LADS A8002.) The P0–3 signal is then inverted and delayed 10 ns to produce the P0–2 signal. The P0–2 signal, in turn, is inverted and delayed to generate P0–1. The remaining scan clock signals are generated in the same manner as P0–2 and P0–1. These clock signals are distributed throughout the scan logic to time scan operations. Clock and not-clock signals have the same relation to scan triggers and latches as CE clock and not-clock signals have to functional logic (see Chapter 2, Section 1).

The advantage of having a separate scan clock for scan logic is that during scan operations the CE clock can be turned off, leaving the scan clock in control. This function is under control of the 'maintenance mode stop clock' (MMSC) trigger. (The operation of the MMSC trigger is discussed in "Scan Stop-CE-Clock Logic".)

With both the scan and CE clocks running, scan hardware is controlled by the scan clock, and normal CE functions are controlled by the CE clock.

FLT Clock Highlights

- Provides scan control timing and counts machine cycles.
- Consists of 'FLT clock-0' and '-1' triggers and 'FLT time-0' to '-3' latches.
- Stepped by scan clock.
- Changes at the fall of P0 (100 time).

The FLT clock (Diagram 6-103, FEMDM) is the prime scan operation sequencer which, in conjunction with the scan clock, provides scan control timing and counts machine cycles. It consists of two triggers, 'FLT clock-0' and '-1', and four latches, 'FLT time-0' to '-3'. Clock and not-clock signals from the scan clock step the FLT clock to provide the outputs shown in the timing chart. Six signals are generated, 'clock-0' and '-1' and 'FLT time-0' to '-3'. The clock steps once for each machine cycle. Thus, in four machine cycles, the FLT clock steps from 'FLT time-0' to '-3' and then recycles.

Two triggers, 'FLT clock 0' and 'FLT clock 1', are stepped by scan clock signals and by the conditions of the four latches: 'FLT time-0', '-1', '-2', and '-3'. The FLT clock triggers function as a reverse binary counter of scan clock cycles. The FLT time latches record the count indicated in the FLT clock triggers.

Assume that the 'FLT time-0' latch is set and both FLT clock triggers are reset (00). Because 'FLT time-1' to '-3' latches are reset, the rise of scan clock P0 sets both FLT clock triggers (11). At not-clock P0 time, the 'FLT time 0' latch is reset, and at not-clock P1 time, the 'FLT time-1' latch is set. Then, at clock P0–1 time, both FLT clock triggers are reset.

Because the 'FLT time 1' latch is set, only the 'FLT clock 0' trigger is set at clock P0 time, at which time the triggers equal 10. This operation continues with the triggers counting down binarily from 11 to 00 during clock time and the latches stepping up to 11 during not-clock time. When the trigger count is 00, the next not-clock P1 signal sets the 'FLT time 0' latch, and the cycle is repeated.

The FLT clock triggers are also used as the low-order two bits of the FLT counter and the cycle counter. As shown in Diagram 6-103, T(62, 63), containing MCW(30, 31), can be transferred to these triggers during cycle counter operation. Operation of the triggers as part of the FLT counter and the cycle counter is discussed in "FLT Counter" and "Cycle Counter", respectively.

Scan Counter Latches and Decrementer

Two counters and two sequencers (Figure 4-3) control the sequential operation of scan-associated functions: the address sequencer, the FLT counter, the ROS test sequencer, and the cycle counter. (The latter is an 11-bit

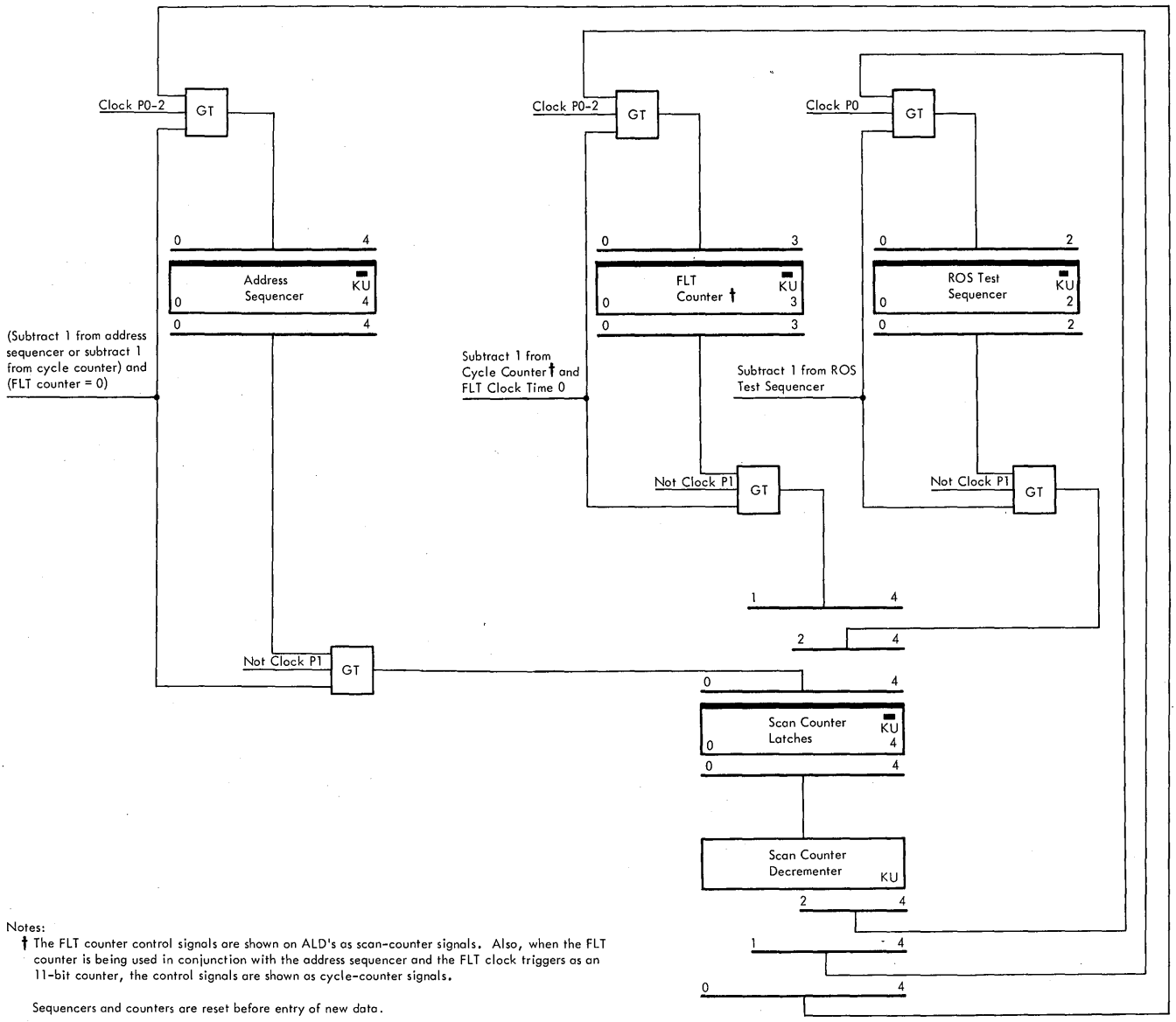


Figure 4-3. Scan Counter Latches and Decrementer Data Flow

counter composed of the address sequencer, the FLT counter, and the two FLT clock triggers.) Each sequencer or counter is decremented by routing the contents of the sequencer or counter through the scan counter latches and back to the source via the scan counter decrementer. The latter subtracts 1 from the value gated through it. However, although all counters are decremented by the scan counter decrementer, only one counter may be stepped during any one clock cycle.

Input and Output

Inputs to the scan counter latches are from the address sequencer, the FLT counter, and the ROS test sequencer. Input is accomplished at not-clock time. The scan counter latch output automatically conditions the scan counter decrementer and, at the following clock time, the decrementer output is transferred into the source counter or sequencer. Inputs to the scan counter latches and to the source counter or sequencer are under control of the same 'not-clock P1' signal as shown in Figure 4-3. This signal resets the counter or sequencer before the decremented value is entered.

Scan Counter Decrementer

The scan counter decrementer (Diagram 6-104, FEMDM) is a logic network that subtracts 1 from any value routed

through it. For example, assume that the address sequencer contains 21 (decimal) or 10101. When the address sequencer contents are to be decremented, they are transferred to the scan counter latches. The value is then routed through the decrementer, where it is reduced to 20 (10100) and sent back to the address sequencer.

Address Sequencer

- Provides data used to generate main storage addresses; generates signals to select scannable triggers and latches on scan operations; functions as five high-order positions of cycle counter.

- Inputs are from T (53–57), hardware (set to 23), ROS micro-orders (set to 7, 13, 15, 16, 23).

The address sequencer, a five-position trigger register, has two main functions: to sequence through a predetermined number of main storage addresses and to select the scan address of the scannable triggers and latches to be scanned out to PAL. When the address sequencer is performing the first function, either a fixed address or a portion of the MCW is placed in it, and its output is transferred to a storage address generator which adds the necessary bits to make up the PSA address. The address sequencer contents are then decremented and transferred to the storage address generator to select the next address, etc. For the second function, the sequence is generally the same, except that the contents are transferred to the address sequencer decoder, which brings up 1 of 24 gating signals to scan out the proper scannable triggers and latches. Decrementing is accomplished by transferring the address sequencer contents to the scan counter latches and back again via the scan counter decrementer.

The address sequencer is also used as the five high-order positions of the 11-bit cycle counter during log-on-count or pulse-mode operation. This function of the address sequencer is discussed under "Cycle Counter".

The address sequencer (Figure 4-4) can be loaded with preassigned values by micro-orders or hardware, or T(53–57) can be transferred to it depending upon the operation. Its contents can be transferred to the address sequencer decoder for selection of scannable triggers and latches, to the storage address generator for addressing storage, or to the scan counter latches and decrementer.

During the scan-in portion of FLTs, the address sequencer is set to a count of 16 and then decremented by 1 for each test word fetched from main storage. The 16th word fetched (address sequencer = 1) is the MCW part of the FLT test pattern and contains a new setting for the address sequencer. This new setting selects the scannable triggers and latches to be scanned out for the test comparison. The scan-out word is selected by decoding the address sequencer contents and the LW bit of the MCW. This action selects the roller switch indicating the 'exit' trigger.

The action taking place while the address sequencer steps from 16 to 0 is under control of the micro-program. When the count in the address sequencer is 0, the scan-in is complete for FLTs.

During a logout operation, the address sequencer is initially set to a count of 23 (decimal) and then decremented by 1 for each doubleword logged out. Again, the count in the address sequencer is decoded by the address sequencer decoder, and the logout word is selected using the indicator drivers as the source point. The address sequencer count is also decoded by the storage address generator to yield a main storage address for the logout word. This address is then put on the SAB before storing the logout word in main storage. While the sequencer steps from 23 to 14, the CE clock is turned off to prevent register operation. At a count of 13 (decimal), the CE clock is again allowed to run, and the logout operation continues under ROS control. At count 0, logout is complete.

During logout, the address sequencer goes to 0 twice. The first time the count goes to 0, the sequencer is forced to a count of 23. This action recalls the first word logged out (word 23, ST contents) to correct possible incorrect parity. The word is then restored, and the sequencer is set to a count of 7 and again allowed to count down to 0, at which time the original parity bits are stored and the operation is completed. (Because no micro-order exists to set the address sequencer to 0, it is set to 7 and allowed to decrement to 0.)

Address Sequencer Decoder

The address sequencer decoder (Figure 4-5) interprets the output of the address sequencer during scan operations. The decoder consists of a high-order and a low-order bit section. The high-order bit section decodes the two high-order bits to yield, for example, 10XXX. The three low-order bits are then decoded to give, for example, xx111. The combinations of the two, in this instance, show the address sequencer count to be 23 (10111). Thus, the word logged out or scanned out would be word 23, the contents of S and T.

The scan logic uses the indicator drivers, normally used for CE control panel display, as scan-out source points. Therefore, the address sequencer selects the roller switch that contains the desired information. In the case where the count of the address sequencer is 23, the desired information is found in rollers 1 and 2, position 3 (S and T) (see "Scan-Out Bus").

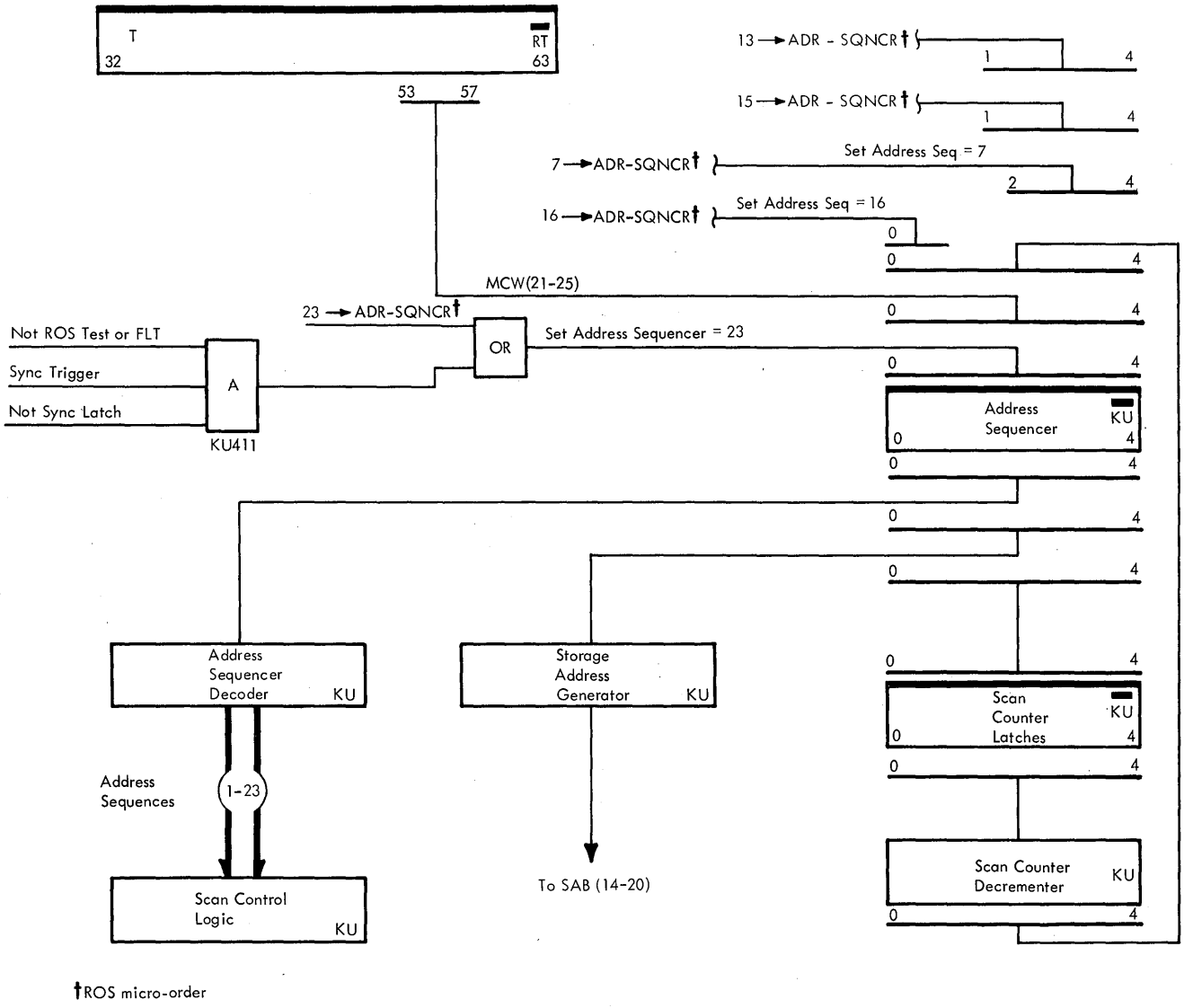


Figure 4-4. Address Sequencer Data Flow

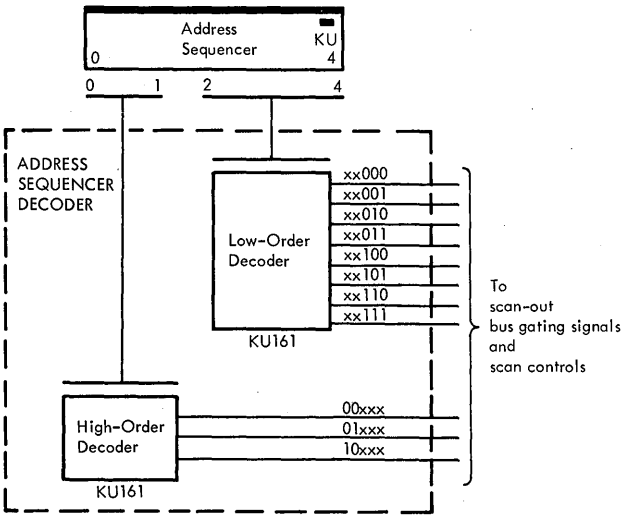


Figure 4-5. Address Sequencer Decoder

Storage Address Generator

- Scan operations use three areas of the PSA: logout area (80–138, hex), buffer 1 (100–180, hex), buffer 2 (200–280, hex).
- SAB values to address location within the three areas are derived from address sequencer contents.

Three areas of the PSA are used for scan operations: locations 80 through 138 (hex) are the logout area, which contains the logwords after a log operation; locations 100 through 180 (hex) and 200 through 280 (hex) are the two areas that contain the FLTs or ROS tests. The SAB values necessary to address any location within these three areas are basically derived from the contents of the address sequencer. Forming a 24-bit address from the five bits of

the address sequencer is accomplished by the storage address generator (Diagram 6-105, FEMDM).

The chart in Diagram 6-105 shows how the storage address is encoded. For all scan operations, bits 0–13 and 21–23 of the storage address are set to 0's (bits 21–23 are 0's because logout words, FLT words, and ROS-test words are doublewords which are always on doubleword boundaries). Address sequencer bits 0 and 1–4 are transferred to SAB (16–20) to select the location within a scan area of storage.

In logout operations, bit 16 of the address is set to the complement of bit 0 of the address sequencer. Thus, for logout area addresses 80 through F8 (hex), SAB(15) is set to 0 and SAB(16) to 1; for addresses 100 through 138 (hex), SAB(15) is set to 1 and SAB(16) to 0.

FLTs or ROS tests are always contained in two areas of storage called buffers. Buffer 1 is from 100 through 180 (hex); buffer 2, from 200 through 280 (hex). When ROS tests or FLTs are being performed, each test, as it is read in, goes to the opposite buffer. For example, if the first test read from tape is placed in buffer 1, the next test is placed in buffer 2, the third in buffer 1, and the fourth in buffer 2, etc. This scheme allows one test to be performed while the next test is being read into storage.

'Buffer 1' trigger controls which buffer is addressed. When 'buffer 1' trigger = 1, buffer 1 is addressed; when 'buffer 1' trigger is off, buffer 2 is addressed. During FLTs and ROS tests, SAB 14 and 15 are under control of the 'buffer 1' trigger. When this trigger is set, indicating that buffer 1 is to be addressed, SAB(15) is forced to 1. When the 'buffer 1' trigger is reset, indicating that buffer 2 is to be addressed, SAB(14) is set to 1.

During ROS tests, the 'buffer 1' trigger is complemented every time the ROS test sequencer equals 7. During FLTs, the 'buffer 1' trigger is under control of the INV-BFR-TGR micro-order. If this micro-order is decoded, the 'buffer 1' trigger is complemented.

SAB gating signals during scan operations are controlled by the 'scan sync' trigger. When this trigger is set, the contents of the address sequencer are transferred onto SAB. During FLTs and logout operations, the trigger is set by the 'ROS MS-REG*SCAN4' micro-order. During ROS tests, it is set when the ROS test sequencer equals 1, 3, or 6.

FLT Counter

The FLT counter is a four-latch register/counter which is coupled with the two triggers of the FLT clock to make a six-position counter. Its main purpose is to count the number of cycles that the CE is allowed to advance after scan-in and before scan-out. During FLTs and ROS tests, the FLT counter is set to the desired count by the cycle count field [MCW(26–31)]. During log-on-count and pulse-mode operations, the FLT counter (four bits) is combined with the address sequencer (five bits) and the two triggers of the FLT clock to make up an 11-bit cycle counter.

The two parts of the FLT counter are decremented differently. The low-order bits, FLT-clock times 1 and 2, are a reverse binary counter that counts down from 3 to 0. Decrementing is accomplished by signals from the scan clock. Each time the low-order bits reach 0, the high-order bits, FLT-counter 0–3, are decremented via the scan counter decrementer in the same manner as the address sequencer and the check counter. The next scan clock signal resets FLT-clock times 1 and 2 to 3.

Input

The FLT counter (Figure 4-6) is loaded from T(58–61) or forced to the maximum count. Except during certain portions of the ROS tests, the MCW cycle count field is gated from T(58–63) to the FLT counter and FLT clock by the 'ROS TB→MCW' micro-order. During ROS tests, the cycle count field is gated from T to the FLT counter when the ROS test sequencer equals 2.

FLT Counter Decrementing

The FLT counter (Diagram 6-106, FEMDM) is decremented under control of two latches: the 'scan counter control' latch and the 'FLT counter equal 0' latch. Initially, the 'FLT counter equal 0' latch is reset, conditioning one leg of AND 1, the 'FLT clock-0' and '-1' triggers are set to some value (normally 3; see "FLT Clock"), and the 'FLT counter-0' to '-3' triggers are set to some value (either to maximum or to the value contained in the MCW cycle count field). When the FLT counter is to be decremented, the 'scan counter control' trigger is set which, in turn, sets the 'scan counter control' latch to bring up a second leg of AND 1.

For FLT operations, the 'scan counter control' trigger is set by the 'ROS 1→CTR CTL TGR' micro-order. For ROS tests, the trigger is set when the ROS test sequencer equals 2. A log-on-count-with-address-compare operation [Diagnose instruction with MCW (1, 6) set] also sets the trigger if the storage address agrees with the ROS address field of the MCW.

With three conditions on 'AND 1' met (i.e., 'scan counter control' latch set, not SOROS and sync latch, and FLT counter not equal to 0), as soon as the FLT clock is decremented to 0 (decremented once per cycle), the FLT counter contents are decremented by transferring them to the scan counter decrementer and back again. At the same time, the FLT clock is reset to 3. When the FLT clock is

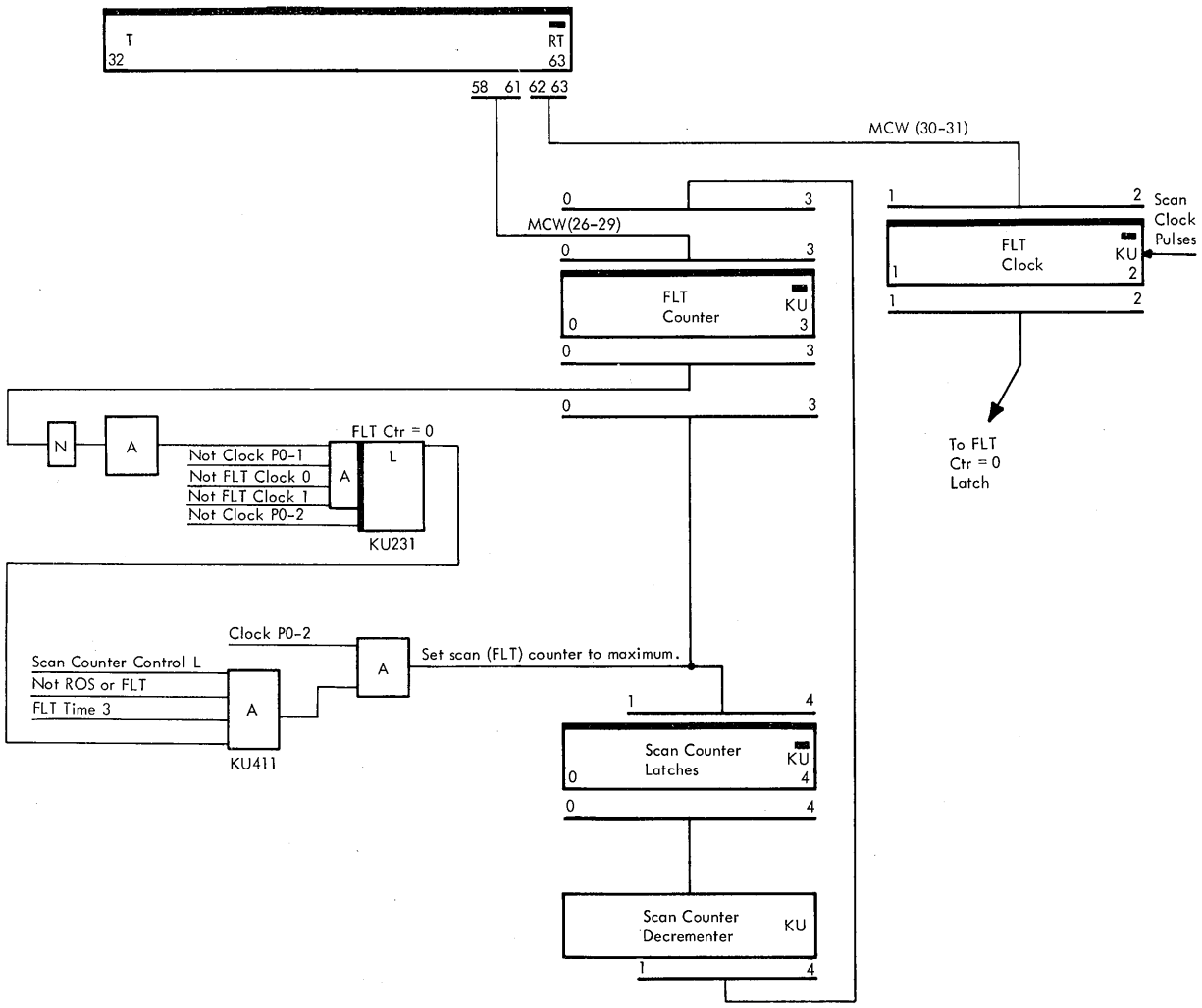


Figure 4-6. FLT Counter Data Flow

again decremented to 0, the FLT counter is again decremented. This operation continues until the FLT counter, including the 'FLT clock-0' and '-1' triggers, is at 0 (reset). At that time, the 'FLT counter equal 0' latch is set, which inhibits further decrementing of the FLT counter.

Cycle Counter

During pulse-mode and log-on-count operations, the address sequencer is joined with the FLT counter and the two triggers of the FLT clock to form an 11-bit cycle counter (Figure 4-7). The low-order positions of the cycle counter are decremented as described under "FLT Counter Decrementing". When the FLT counter is decremented to 0 ('FLT counter equal 0' latch set), the address sequencer contents are decremented (gated to scan counter decrementer and returned to address sequencer). At the same time, the 'FLT clock-0' and '-1' triggers are set, and the FLT counter is set to the maximum (all 1's). These counters are decremented as before until they reach 0 again; the address sequencer is then decremented, and the operation is repeated.

When all three counters equal 0, the 'cycle counter equals zero' signal is generated to stop the clock (pulse-mode) or to perform a logout operation.

ROS Test Sequencer

The ROS test sequencer (Figure 4-8), three polarity-hold circuits, controls the scan logic during a ROS test. At the start of the ROS test routine, the ROS test sequencer is set to a count of 7; it is decremented at FLT-time 3 by having its contents transferred to the scan counter latches and decrementer. The output is decoded by the ROS test decoder, which generates one of seven signals, depending upon the present value of the sequencer. These signals control the sequential operation of a ROS test.

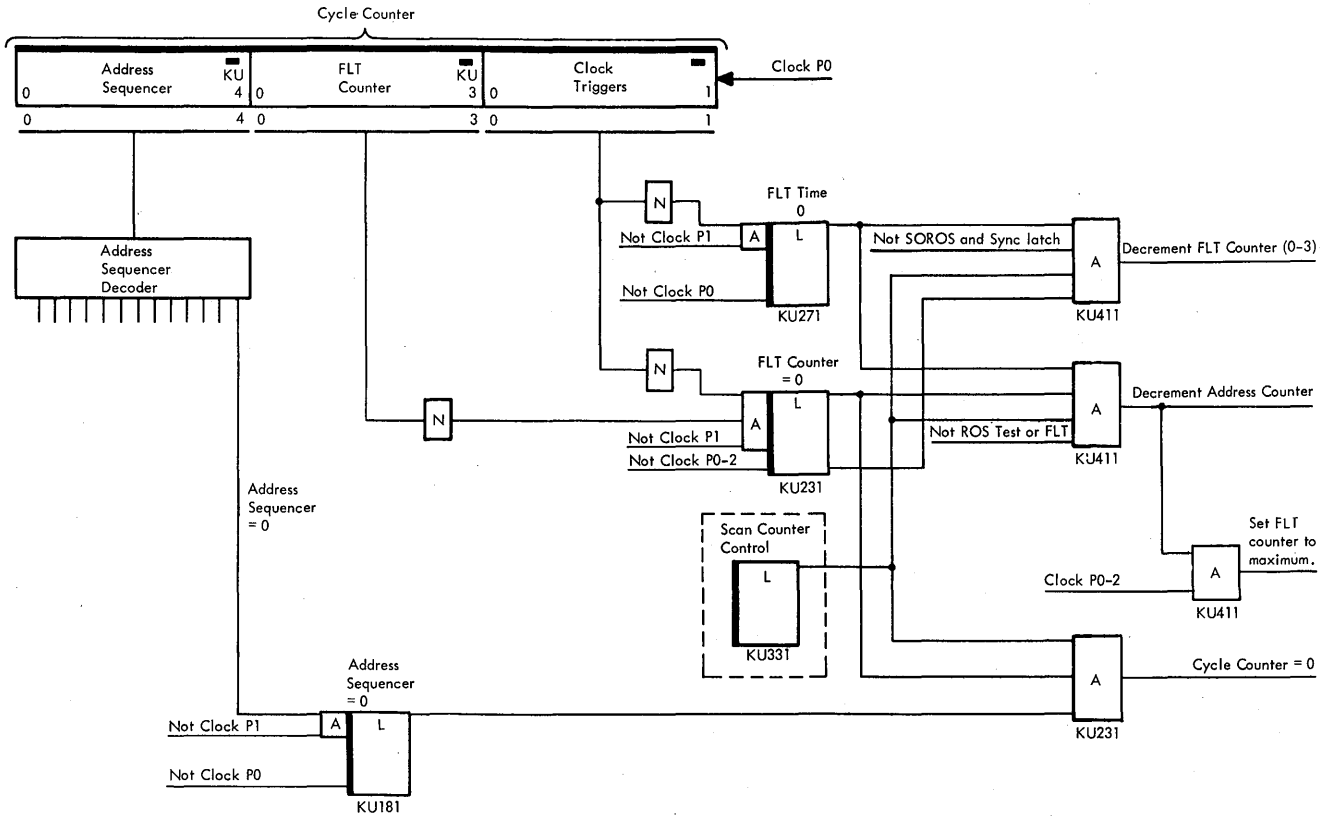


Figure 4-7. Cycle Counter Data Flow

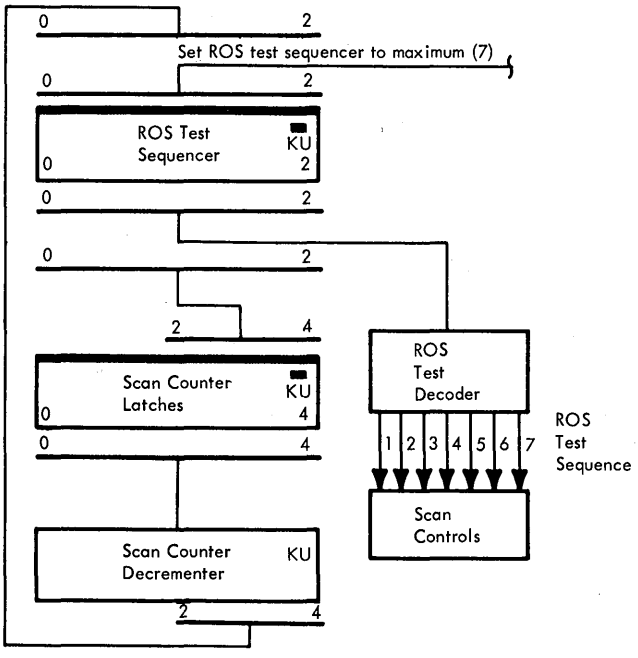


Figure 4-8. ROS Test Sequencer Data Flow

Scan-Out Bus

- Scan-out bus is data path from indicator logic to PAL.
- Status of 64 indicatable storage devices (scan-out word) is scanned out to PAL by one scan-out address.
- One-half of scan-out word is transferred to PAL during one cycle.

The scan-out bus (Diagram 6-107, FEMDM) is a special data path used in scan operations that allows direct transfer into PAL from storage devices (triggers, latches, polarity-holds) which ordinarily have no direct input into PAL. The operation that performs this function is referred to as "scan out". Scan out makes use of the indicator roller switch position logic to perform this transfer. Thus, the first requirement necessary to scan out a storage device (except for the LSWR) is that the device be indicatable; that is, that it has an indicator on the CE control panel roller switches.

During a scan-out, the status of 64 indicatable storage devices is scanned out using one scan address (address sequencer contents); the 64 bits transferred to PAL are

referred to as a scan-out word. The 24 scan-out words are listed on LADS A6521-A6561 as logout words because, in a logout operation, the scan-out words are logged out (placed in the logout locations of the PSA). Note that there is no direct correlation between the scan-out address and the roller switch position of the indicator. For example, in Diagram 6-107, scan-out address 23 is shown as selecting the same storage device as the 'roller 1 position 3' signal from the CE control panel.

The indicatable storage devices to be scanned out at any one time are selected by the output of the address sequencer decoder and by a signal from the scan controls. Only one-half of the scan word is transferred to PAL during one cycle; therefore, the signal from the scan controls will be either 'scan out left' (selects bits 0–31 of a scan word) or 'scan out right' (selects bits 32–63). The output of the address sequencer decoder and the 'scan out left' or 'scan out right' signal places a signal on the same line as does the CE control panel roller switch (Diagram 6-107). This signal allows the storage device output signal to light an indicator on the system control panel or to be transferred to PAL via the scan logic.

At the same time, to prevent the roller switch setting from affecting the scan-out, the 'not block indicator switches' signal is activated. Thus, the only indicatable storage devices whose outputs are sent to the indicator drivers are those selected by the address sequencer decoder output and the scan-out signal.

Gating signals route the indicator signals from the storage devices to two places: to indicators on the CE control panel and to the scan-out bus. From the scan-out bus, they are transferred by the 'enable bus' signal to PAL(32–63).

Logout Controls

The controls for a logout operation are shown in Diagram 6-108, FEMDM. A logout is started by an 'error log required' signal, by depressing the LOG OUT pushbutton (in manual mode only), or by executing the Diagnose instruction. The 'SOROS' trigger sets the address sequencer to 23. The 'SOROS' trigger and the 'sync' latch request a storage cycle to store the logged-out word. The FLT clock provides timing signals to control the logout operation until the address sequencer reaches 14; the CE clock is then started and ROSAR is forced to 019 (hex) to control the reset of the logout operation.

Diagram 6-109, FEMDM, illustrates the scan-out path to PAL(54) on a logout operation. Assume that the address sequencer equals 10 (decimal). A scan-out is initiated to transfer the contents of D(22) to PAL(54) if the 'scan out right' signal is generated. Normally, the 'scan out right' signal is generated by the 'SCAN OUT-RTWD' micro-order. However, on a scan-out ROS operation, it is generated by the scan controls. At the same time that the 'scan out right' signal is generated, the 'enable scan bypass' signal is produced. This signal is generated by the 'SCAN BYPASS' micro-order or, in the case of a scan-out ROS operation, by the scan controls. In addition to allowing transfer into PAL, the 'enable scan bypass' signal inhibits transfer to the CE control panel indicators by the switchable indicator logic.

With the 'scan out right' signal active and the address sequencer equal to 10 (decimal), D(14) is transferred to indicator 25 of the row of indicators normally selected by roller 1 and to the scan-out bus. Because the 'scan out S and T' signal is not up at this time, D(14) is transferred to PAL(54) by the 'enable scan bypass' signal. At the same time, the remainder of the right half of scan word 10 is transferred to the appropriate bits of PAL.

Scan Out S and T

This operation gates both the S-register and the T-register into the PAL. One bits from either register result in 0 bits at the output of the PAL because of inversion. This function is used to examine the state of a trigger to be tested in ROS or FLTs. It also performs the comparison required in an alternate-test-number search. To test a trigger, the address sequencer selects a 64-bit logout word. The 'Left Half' bit in the MCW, bit 4, selects which 32-bit half contains the trigger to be tested. These 32 bits are placed in the T-register and the Mask is placed in the S-register. After scan-out S and T, the PAL output is 0 if the trigger was set and non-0 if it was reset.

The 'scan out S and T' signal is a result of the 'SCAN OUT S-REG' and 'SCAN OUT T-REG' micro-orders or the ROS test sequencer equalling 1 or 4.

Scan Stop-CE-Clock Logic

During certain scan operations, the CE clock must be stopped while the scan clock is allowed to run. This operation is controlled by the 'MMSC' trigger (Diagram 6-110, FEMDM). When this trigger is set, the unsymmetrical clock signal that controls CE trigger and latch setting and resetting is inhibited; thus, all CE functions controlled by CE clock signals are stopped, except as allowed by scan clock signals.

The 'MMSC' trigger is set by the following conditions:

1. Console LOGOUT pushbutton.
2. External CE logout request.
3. Pulsed split log (invalid address or SE timeout during CE logout).
4. During FLT or ROS tests.

Control Triggers

The scan logic contains many triggers which are used for status and control. The most important of these, with their functions, are:

1. 'FLT Backspace' (Diagram 6-27). Set whenever BACKSPACE FLT is depressed during FLT or ROS testing. This trigger conditions Subsystem Reset and raises 'FLT BACKSPACE IOCE (x)' and 'FLT LOAD'. The test tape is backspaced one record after which the trigger is reset.
2. 'Scan mode'. Set when the scan controls are under microprogram control. The ROS fields used by the scan microprogram are shared by the non-scan functions of the CE. They are interpreted as scan micro-orders if the 'scan mode' trigger is set.
3. 'Sync' (Diagram 6-111). Starts a ROS test or an FLT after an IPL operation by synchronizing these operations with the scan controls.
4. 'Repeat'. Scan mode REPEAT TEST switch. When set, the ROS test or FLT currently running is repeated continuously.
5. 'FLT test'. Set by the SCAN MODE, ROS/PROC/FLT switch being in the FLT position. It places the CE in FLT mode.
6. 'Scan out ROS' (SOROS) (Diagram 6-108). Set by a machine error when in log-on-error mode, by the LOG OUT pushbutton, or by the 'cycle counter = 0' signal when performing a log-on-count operation. External CE logout required or Pulsed Split log (SE error during CE logout). Initiates the scan-out portion of a logout operation.
7. 'Pass', 'fail' (Diagram 6-111). Store the results of a ROS test or FLT. MCW(7) is the ERSLT bit, and PAL equals 0 if the bit to be tested in the scan word is a 1. Thus, if MCW(7) equals 1 and PAL equals 0 or if MCW(7) equals 0 and PAL is not equal to 0, the 'pass' trigger is set. If MCW(7) equals 1 and PAL is not equal to 0 or if MCW(7) equals 0 and PAL is equal to 0, the 'fail' trigger is set. For FLTs, the test is performed under microprogram control. For ROS tests, the test is performed when the ROS test sequencer equals 1. In either case, if both the 'pass' and 'fail' triggers are set, an intermittent error occurred.

Scan Mode Control of ROS

Scan mode operations affect three fields of ROSDR: field D (bits 17–19), field F (bits 25–30), and field G (bits 31–35). These fields serve dual functions. In the normal mode, they are decoded from the ROSDR latches as standard CE control lines. In scan mode, they are decoded as special scan control lines and are referred to as field S (LADS A6261).

The scan mode is controlled by the 'scan mode' trigger. When this trigger is reset, the standard decode path is used. When this trigger is set, however, the standard control lines are blocked and scan control lines (using common CE control line codes) are activated.

The scan control logic generates blocking signals to inhibit register gating signals at the ROSDR decode logic and to allow scan control use of ROS in sequencing through its test operations. Scan logic also affects ROS micro-branching (Diagram 6-112, FEMDM). The J-field micro-orders shown in the diagram are listed on LADS A6231.

CE Scan/IOCE Interface

The CE's actions in an FLT or ROS IPL are different from a normal subsystem IPL in that hardware rather than ROS determines when the unit select switches and 'IPL IOCE (X)' are gated. Also, the CE uses 'TIC' and 'GAP' rather than 'Response' from the IOCE to determine the IOCE's progress in reading data into storage (Diagram 6-113, FEMDM). The following CE-IOCE interface lines are of particular interest in an FLT or ROS IPL:

1. 'TIC' (IOCE to CE). This signal is sent to the CE when the IOCE decodes a CCW specifying transfer in channel. SEL CH REQ TIC RTNE-KK631 in the IOCE brings up 'TIC'. It informs the CE that a buffer is loaded and free to be fetched.
2. 'GAP' (IOCE to CE). This signal is sent to the CE when the IOCE encounters an IRG, unless the channel is executing a CCW specifying command chaining. 'GAP' is called EOR in the IOCE. It is brought up by SET PCI REQUEST.
3. 'FLT Backspace' (CE to IOCE). This signal is sent to the IOCE in parallel with 'FLT load' for an FLT backspace operation. In this case, the IOCE's hardware IPL CCW specifies backspace.
4. 'FLT Load' (CE to IOCE). This signal is not used in the 9020D or E for an FLT or ROS IPL. The normal signal, 'IPL IOCE (X)', is used instead. See 'FLT backspace'.
5. 'Response' (IOCE to CE). In a normal IPL, this signal informs the CE that an IPL operation has been completed; a CCW specifying neither 'TIC' nor 'chain' was encountered. In an FLT or a ROS IPL, the IOCE is in a CCW loop. When it reads an IRG, it gets reset to the halt loop. No 'response' signal is sent because no 'non-TIC' or 'chain CCW' is encountered.
6. 'FLT Load Complete' (IOCE to CE). This line is not used in the 9020D or E when loading. It is the response to a backspace signal to notify the CE that the backspace is complete.

LOGOUT

Introduction

The logout function of the scan logic stores the status of various triggers and registers, reflecting the state of the CE, into predesignated locations of the PSA area. The 24 doublewords logged out are stored in PSA locations 80 through 138 (hex). (For a detailed list of the 24 doublewords, refer to LADS A6521–A6561 or Diagram 6-117, FEMDM.)

The status of each trigger logged out is represented by a 1 if it is set or by a 0 if it is reset. Thus, a record of the machine state, at the time that a CE or storage error occurs, is stored unchanged in predetermined locations of the PSA area with a fixed format. This record can then be accessed by a program or by manual controls for analysis, printout, or display.

A logout operation can be initiated by:

1. Manually depressing the LOG OUT pushbutton on the CE control panel. The system must be in CE Control (mode) (Diagram 4-210, FEMDM).
2. Executing the Diagnose instruction when a log-on-count function is specified. Logout occurs after a predetermined number of CE clock cycles (preset by the diagnostic programmer).
3. Detecting a machine check during normal CE operation if the CHECK CONTROL switch is in the PROC position and the PSW machine check mask bit is on.

Operational Analysis

- Logout stores 24 doublewords, which reflect CE status, in PSA locations 80–138 (hex).
- Operation is controlled by scan logic and ROS microprogram.
- Address sequencer is set to 23 and decremented by 1 for each logword stored.

The logout function, whether initiated by depressing the LOG OUT pushbutton, by executing the Diagnose instruction, or by detecting a machine check, stores 24 doublewords (log words), which reflect the CE status, in PSA locations 80–138 (hex). The information presented in these logwords is determined by the scan logic and is therefore fixed. A list of the logword locations and their contents is presented in Diagram 6-117 and LADS A6521–A6621.

The words are logged out in a fixed sequence, as defined by the scan logic, with logword 23 being stored in location 238 (hex), logword 20 in location 120, etc., in reverse order, until logword 0 is stored in location 80 (hex).

The logout operation is both hardware- and ROS-controlled. When a logout operation is initiated, for example, by depressing the LOG OUT pushbutton, the address sequencer is set to 23 and is decremented by 1 for each logword stored. From the instant that LOG OUT is depressed until the address sequencer is reduced to 14, the scan hardware controls the operation. When the address sequencer equals 13, control is switched to a ROS microprogram which completes the logout.

Although the CE is under ROS control for the last 14 words logged out, the address sequencer is still decremented for each logword. This decrementing occurs because the address sequencer, in addition to controlling the logout sequence, defines the storage address for each logword and the indicatable storage devices to be logged out (Diagram 6-105).

The logwords compose the status of most indicatable storage devices (i.e., triggers, latches, and registers that have an indicator on the roller switches). For this reason, the scan-out bus, which is a data path from the indicator drivers to PAL, is used to transfer the status of these devices to PAL (the path used is identical with that used in an FLT or ROS test during a scan-out; see "Scan-Out Bus"). From PAL, the logwords are gated to ST and then to main storage via the SDBI. Correct parity is assigned in PAL.

Because cycling of the CE clock could change the contents of some of the storage devices logged out, the CE clock is turned off (blocked) while logwords 23 through 14 are being logged out, and timing is controlled by the scan and FLT clocks. Scan/FLT clock signals are distributed throughout the scan logic to control the logout function during the period that the CE clock is turned off. For example, the address sequencer is decremented at FLT-time 3, a latched output of 190-ns duration.

At the end of a logout, the CE performs an end op, and the 'machine check interrupt' trigger is set.

(A Storage or Display Element logout is initiated only by programming, using the Diagnose instruction and Diagnose Kernel FDO. Six (logout) doublewords are stored in the locations immediately following the MCW specifying the FDO kernel. The formats of these logwords are shown in Diagrams 6-118 and 119, FEMDM.)

Hardware-Controlled Sequence

- Logout is initiated by Diagnose instruction, LOG OUT pushbutton, machine error, or External CE Logout request.
- 'SOROS' trigger initiates hardware-controlled portion of logout sequence.
- For each word logged out: (1) address sequencer is decremented, (2) scan-out logic places right half of

logword into T, and (3) logword is stored in main storage location addressed by storage address generator.

- Only right half of logwords 23–14 are logged by the hardware sequence. The left halves of 23–18 are logged by the ROS sequence.

As stated above, a logout (Diagram 6-114, FEMDM) may be initiated by executing the Diagnose instruction, by depressing LOG OUT, by detecting a machine error when in log-on-error mode, or by External CE Logout request. The operational differences between the methods occur before the actual logout sequence and are as follows:

1. The Diagnose instruction initiates a logout operation if MCW(6) = 1, thus specifying a log-on-count operation. In this case, MCW(21–31) is sent to the cycle counter, which is decremented by 1 each machine cycle. When the cycle counter equals 0, the 'SOROS' trigger is set, initiating a logout sequence.
2. If the logout operation is initiated by depressing LOG OUT with the CE in manual mode, the 'console logout' latch and the 'pass pulse' trigger are set. Normally, the 'pass pulse' trigger will already be set. However, if the RATE switch is in the SINGLE CYCLE position, the 'pass pulse' trigger remains set only for the duration of one CE clock signal (Diagram 4-1, FEMDM). This action allows the logout operation to be stepped through one cycle at a time by depressing START for each cycle. With the 'pass pulse' trigger and 'console logout' latch both set, the 'SOROS' trigger is set.
3. When the CE CHECK switch is in the PROC position and the PSW machine check mask bit is a 1, a machine check ('error' trigger set) initiates a logout. Any of the Check Reg 1 or 2 (except hardstop) errors (Diagram 6-22) sets the 'error' trigger which, in turn, sets the 'SOROS' trigger.
4. A WDD (External CE) logout request initiates a logout by setting the 'SOROS' trigger directly.

Note that, in each case, the 'SOROS' trigger is set (Diagram 6-108), thus starting the hardware portion of the logout sequence (Diagram 6-114). After the 'SOROS' trigger is set, the 'sync' trigger is set at FLT-time 3 to synchronize the logout operation to the FLT clock. The output of the 'sync' trigger sets the address sequencer to 23 (first logword) and sets the 'sync' latch at not-clock time. (The 'sync' latch being set causes the address sequencer to be decremented every FLT-time 3.)

With the 'sync' latch set, a storage request to the SCI is initiated at FLT-time 0. This request gates the address generated by the storage address generator to SCI. Because the first word to be logged out reflects the contents of ST (logword 23), these contents are now stored (gated to SDBI).

Since ST may contain bad parity, a 'suppress log check' signal is sent to the SE when storing logword 23 and, also, when later fetching it to assign good parity. The signal tells the SE to inhibit its data parity check.

When the 'sync' trigger was set, the 'MMSC' trigger was also set. This action keeps the CE clock off (the 'error' trigger inhibited CE clock signals) but allows the scan clock to run. Therefore, the status of all operational registers, except those used for the logout (S, T, and PAL), is preserved during the logout sequence.

Because the contents of ST have been stored, the address sequencer is decremented to 22 at FLT-time 3. A scan-out operation is performed, which gates the right half of logword 22 (selected by the address sequencer decoder) to PAL(32–63) from the indicatable storage devices via the scan-out bus. From PAL, logword 22 is gated to T for transfer to main storage. At FLT-time 0, another storage request is issued to store the contents of ST. This operation continues for logwords 19 through 14; that is, for each word logged out: (1) the address sequencer is decremented, (2) a scan-out places the right half of the logword in PAL, which is subsequently gated to T, and (3) the logword is stored in the main storage location addressed by the storage address generator. Note that only the right half of logwords 23 through 14 is stored; the left halves of 17–14 in storage remain unchanged and could contain anything. The left halves of 23–18 are scanned and stored by the ROS-controlled sequence.

When the address sequencer has been decremented to 14 and the logword has been stored, the 'MMSC' trigger is reset to start the CE clock, address 19 is forced into ROSAR, and the address sequencer is reduced to 13. (Note that the ROSAR contents have been already logged out.) At this point, control is transferred to a ROS microprogram.

ROS-Controlled Sequence

- ROS microprogram directs logout from time address sequencer equals 13 until end of logout sequence.

- Logword 13 is formed by: (1) transferring LSWR to S, (2) scanning out to T, (3) transferring T to LSWR, (4) transferring S to T, (5) transferring LSWR to S, (6) storing logword 13.

- Remaining words are formed by: (1) decrementing address sequencer by 1, (2) scanning left word out to T, (3) transferring T to LSWR, (4) scanning right word out to T, (5) transferring LSWR to S, (6) storing logword in storage.

- When address sequencer equals 0, it is again set to 23, parity is corrected on logword 23 (ST), and left halves of logwords 23–18 are scanned and stored.

The ROS microprogram directs the logout operation from the time the address sequencer equals 13 until the end of the logout sequence (Diagram 6-114). The first micro-order in the logout microprogram transfers the contents of the LSWR to S, thus preserving these contents in a logword because the LSWR is used during the remainder of the operation. S now contains half of logword 13. The complete logword is formed by: (1) scanning to T (scan-out to PAL whose contents are transferred to T), (2) transferring the data in T to the LSWR, (3) transferring the contents of S to T, and (4) transferring the contents of the LSWR to S. As a result, ST contains the 13th logword. The operation continues in the following cycle:

1. Decrement address sequencer by 1.
2. Scan out to T (left half).
3. Transfer contents of T to LSWR.
4. Scan out to T (right half).
5. Transfer contents of LSWR to S.
6. Store word into main storage.

This cycle is repeated until the address sequencer equals 0, at which time 24 logwords have been stored in main storage locations 80–138 (hex).

When the logout operation was started, the first word stored in main storage (logword 23) was composed of the contents of ST. The parity associated with this data is not always correct. Therefore, to ensure that correct parity is stored with logword 23, the address sequencer is set to 23, and a 'scan storage request' is initiated to fetch the original contents of ST in logword 23. When available on the SDBO, the word is gated to AB. The address sequencer is then set to 7, and AB (scan address 7) is scanned out. When AB is scanned out, correct parity is inserted for the data (original ST contents). The address sequencer is again set to 23, and a 'scan storage request' is initiated to return logword 23 to main storage with correct parity. Then the sequencer is stepped through 22–18 as the left halves of those logwords are stored. Next, the address sequencer is reset to 7 and repeatedly decremented by 1 until it equals 0, at which time the original ST parity bits are stored. The scan system is then reset, the 'machine check interrupt' trigger is set, and the operation concludes by entering the machine check interrupt routine.

ROS TESTS

Introduction

- Test each bit of each ROS word.
- Tests are on tapes.

ROS tests are the principal means of testing the validity of the ROS bit planes. These tests, generated by a computer program from the tapes used in the manufacture of the ROS bit planes, are stored on magnetic tape. When testing ROS, the tests are read into two PSA buffer areas, starting at locations 100 and 200 (hex). Under scan logic control, the ROS tests compare the value of a particular bit in a selected ROS word with its expected value as specified on the ROS test tape. Each ROS test is read into the CE and checks one bit of a ROS word. At the same time that the test from one buffer area is being executed in the CE, the other buffer area is being filled from the test tape via the IOCE.

The ROS test format consists of two doublewords: word 0 and word 1. Word 1 contains the mask and MCW. The mask, a 32-bit field, in conjunction with the scan-out address field of the MCW, selects the ROS bit to be tested from the word read out of ROS. A particular bit is tested by making all mask bits 1's except the bit that corresponds to the test bit. Word 0 contains the test number (TN) and alternate test number (ATN). The TN, a four-byte field, contains two two-byte numbers that identify the test pattern. The lower-order two bytes are the complement of the TN, and the high-order two bytes are the TN. The ATN, which refers to another ROS test, is also represented in true and complement form, with the complement and true numbers reversed from that of the TN format. The TN refers to the test being executed. The ATN refers to the test that will be executed if the tests are restarted after a failure stop (generally, it refers to the next test).

The first part of each ROS test tape contains hardcore tests to establish that the CE is able to run ROS tests. Testing should not proceed beyond the hardcore tests if failures are encountered.

Following the hardcore tests are the actual ROS tests. During these tests, the ROS word to be tested is selected by the ROS address in the MCW. The CE clock is allowed to generate clock signals to cycle ROS so that the bit under test is placed in the ROSDR. The word containing this bit is defined by the scan-out address in the MCW and is transferred (scanned out) via the indicator driver logic and PAL to the T-register.

The mask is transferred from main storage to the S-register, and then the status of the bit under test is determined by comparing S with T. The result of this comparison is compared with the ERSLT bit to determine whether the test passed or failed. Pass, fail, and intermittent-fail are the three possible results of the comparison. The CT and UT bits in the MCW then determine whether to proceed with testing or to terminate.

If testing is terminated on a failure, the bit plane is displayed as the CE TEST ADDRESS on roller 5, position 2, bits 0–3. The S-register, S(0–7), indicates the failing

word in hex and S(8–15) indicates the failing bit in decimal (Roller 1, pos 3). A nonfailing test is repeated until a 'transfer in channel' (TIC) pulse notifies the CE that the alternate buffer is filled and new test data is available.

Because FLTs cannot be run if malfunctions exist in ROS, the ROS tests should be run first, followed by the FLTs.

Operational Analysis

- ROS tests consist of two doublewords: word 0 has TN and ATN; word 1 has mask and MCW.
- All bits of ROS bit planes are checked for 1 or 0.
- Each ROS test is repeated until receipt of 'TIC' pulse.

ROS tests are scan-controlled tests of the ROS microprogram. Each ROS test consists of two doublewords, designated words 0 and 1, which are read into the CE from the ROS test tape. Word 0 contains the TN and ATN, and word 1 contains a mask and the MCW. A single ROS test checks one bit position of one ROS word and the cycling of data from ROS to the ROSDR.

Each ROS test tape begins with hardcore tests to check out the hardware that controls subsequent ROS tests. Upon successful completion of these hardcore tests, the true ROS tests are begun and continue until all bits of the ROS bit planes have been checked for a 1 or 0 or until an error is encountered. If an error occurs, the CE stops and the failing bit number is displayed in S. Thus, no documentation is required.

The tests are loaded into buffer areas 1 and 2, and 'TIC' pulses are generated as the buffers are filled. Each ROS test is continuously repeated until the CE receives the 'TIC' pulse. Testing then continues if 'pass' is set and 'fail' has not been set.

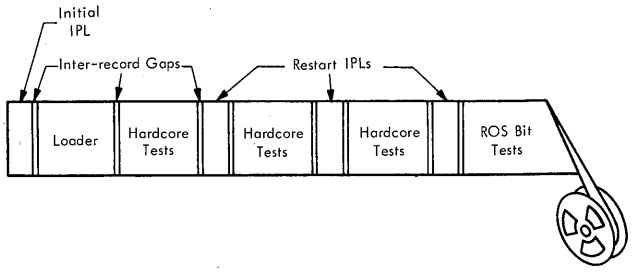
A single test consists of fetching the MCW from storage, permitting the CE clock to advance a given number of cycles, and comparing one of the ROSDR triggers with an expected result. If the state of the trigger matches its predicted value, the 'pass' trigger is set. If the actual and predicted values disagree, the 'fail' trigger is set.

The ROS tests are controlled by the ROS test sequencer which is stepped by the scan clock. At the start of the ROS test, the ROS test sequencer is set to 7 and is decremented to 0 as the test progresses to conclusion. For each count of the test sequencer, a certain part of the ROS test is performed.

The number of clock signals required to move the selected ROS word into the ROSDR is specified in the cycle count field of the MCW. This count is set into the FLT counter; when the counter equals 0, the CE clock is stopped and the bit comparison begins.

ROS Test Tape

The ROS test tape contains the following records:



1. Record 1, Initial IPL. Contains the 24-byte 'bootstrap' program necessary for any IPL operation. When LOAD is depressed, the three doublewords of this record are read into storage locations, as follows:

Storage Location (Hex)	Word	Contents
0		Loader ID
8	CCW 1	Read command to read 8 bytes to location 0 and chain command to location 10.
10	CCW 2	Read command, 28 (hex) bytes into 18 (hex). Chain to 18.

2. Record 2, Loader. Contains the "loader" program that reads in the ROS tests. The IPL program in record 1 reads this record into storage locations as follows:

Storage Location (Hex)	Word	Contents
18	CCW 1	Read 8 bytes into location 0 and chain data.
20	CCW 2	Read command to read 10 (hex) bytes to buffer 1 (location 100, hex) and a chain data tag to location 28.
28	CCW 3	TIC command to location 30.
30	CCW 4	Read command to read 10 bytes to buffer 2 (location 200) and a chain data tag to location 38.
38	CCW 5	TIC command to location 20.

3. Record 3, Hardcore Test 1 and 2. Contains the first ROS hardcore test, two doublewords. Hardcore tests check the scan and CE hardware required to do the actual ROS testing. Any failure encountered during the hardcore tests must be corrected before the actual tests have validity.
4. Record 4, (Restart IPL). The first hardcore causes stop. When LOAD is depressed, the following IPL program replaces the IPL 1 program:

Storage Location (Hex)	Word	Contents
0		Loader ID.
8	CCW 1	No-op. Chain command.
10	CCW 2	No-op. Chain command.

Restart IPLs preceed every test record except the first.

5. Records 5–8, Hardcore Tests. Contain the remaining hardcore tests.
6. ROS Bit Tests. The remaining records on the tape contain the true ROS tests. Each test pattern (two doublewords) tests one bit of one ROS word.

ROS Test Setup

Several controls on the system control panel must be operated to initiate the ROS tests. The procedure to run a ROS test appears in LADS A6503. However, a short discussion of the setup is included here because it affects the operation.

Diagram 6-115, FEMDM, shows the start of a ROS test. The ROS test tape is mounted first. A subsystem must be configured, the CE must be in state zero, with test switch on and an SE selected. The LOAD UNIT switches are set to the address of the tape unit holding the test tape, the SCAN MODE, ROS/PROC/FLT switch is set to the ROS position, and the CE CHECK CONTROL switch is set to DSBL. Going into ROS test mode causes the 'ROS test' latch to be set and a subsystem reset signal to be sent to the IOCE to prepare for a read operation. The manual control operations necessary to get started are concluded by depressing SYSTEM RESET, setting all DATA switches to 1's, depressing STORE to transfer the DATA switches to ST, and depressing LOAD.

Initial IPL Highlights

- IPL is under hardware control.
- IOCE operations are same as normal IPL.
- CE clock is stopped until release is received from channel.

Depressing LOAD with the TEST MODE, ROS/PROC/FLT switch set to ROS initiates an IPL operation that is different from the normal program load in that the operation is under hardware control (Diagram 6-115, FEMDM). IOCE operations are identical with a normal IPL operation; 24 bytes are read from the selected device into the first three doubleword locations of the PSA area. However, when the 'IPL status' trigger is set, the 'MMSC' trigger is also set, which stops the CE clock. Because the CE clock is stopped, the IPL microprogram is not initiated and the remainder of the IPL is under scan and IOCE control.

At this point, the CE is idle, waiting for the first 'TIC' signal from the IOCE; only the scan clock is running. Meanwhile, the IOCE IPL operation reads in record 1 of the ROS test tape and executes the channel program specified by record 1. As a result, 40 bytes (record 2) are read into storage starting at location 18 (hex). Record 2 contains the loader program that reads each ROS test into the proper buffer area in storage.

After record 2 has been read in, command chaining causes CCW 2 in record 1 to be executed. CCW 2 is a TIC command to location 20, which now contains a read CCW. When the IOCE executes the TIC command, it sends a 'TIC' signal to the CE, which is the release signal the CE is waiting for.

Loader

While the scan controls are being set up to run the first hardcore test, the TIC command in CCW 2 of the Initial IPL program causes the channel program in record 2 of the ROS test tape [now in the 40-byte locations of storage beginning at 18 (hex)] to be executed. The read CCWs in locations 20 and 30 cause the channel to read the hardcore tests in record 3 into buffers 1 and 2 (100 and 200, respectively).

Meanwhile, scan control operations in the CE begin when the ROS test sequencer is set to maximum, thus placing the CE in ROS test state 7. In this state, the 'buffer 1' trigger is inverted, the 'Start ROS test' trigger is set, and a scan system reset clears the scan IPL controls. The ROS test sequencer is decremented by 1. During ROS test state 6, an address is forced to storage controls, and a scan storage request for the ROS test word containing the TN and the ATN word is initiated. The operation then waits for a 'TIC' pulse before progressing, but the 'TIC' latch has not been reset since the first 'release' TIC. This 'TIC' pulse results when the first hardcore test in record 3 has been read into storage from the channel (execution of TIC CCW in location 28). After the 'TIC' pulse, the ROS test sequencer is decremented by 1.

At this time, an ATN search is made. The test number, all 1's, from the test buffer is transferred to the S-register, and an equal comparison should result.

Hardcore Test

- Hardcore tests at beginning of FLT and ROS test tapes determine that CE hardware used to run tests is functioning.
- Failing tests stop, with a number in MCW 0–3.
- If hardcore tests run successfully, testing terminates on correct stops when testing is resumed, CE enters true ROS tests or FLT zero-cycle tests.

Every FLT and ROS test tape begins with hardcore tests (LADS A6503) to determine that the CE hardware used in running these tests is functioning properly. These tests are almost identical for ROS tests and FLTs; differences will be signified. Because the CE hardware most involved in running FLTs or ROS tests is S, T, PAL, and the connnecting paths between these points, these are the logic areas checked by the hardcore tests.

These tests include sensing for 1's or 0's in S and T, verifying the CE's ability to stop when the stop conditions are met, and verifying the ability of the CE to conduct an ATN search and sequential testing. A failing test stops testing, with the failing test bit identification pattern displayed in MCW 0–3. If all hardcore tests are run successfully, testing terminates on three hardcore stops (correct stops); when testing is restarted, the CE enters the true ROS tests or, in the case of FLTs, the zero-cycle tests.

Because the ROS hardcore tests are sequenced by the same controls that sequence the ROS tests and because the FLT hardcore tests are executed in the same manner as the regular FLTs, the hardcore tests have a different format for ROS tests and FLTs. ROS hardcore tests have the following format:

		Buffer 1	Buffer 2
TN	ATN	100	200
Mask	MCW	108	208

Each FLT hardcore test contains the 17 words used in an FLT; however, only two of the words are significant for the test. These are words 16 and 1, the TN/ATN and the MCW, respectively. The TN/ATN causes the CE to progress through the tape records. The MCW causes the CE to make decisions. The other 15 words read in during a hardcore test contain 0's and contribute nothing to the test.

Hardcore tests check S, T, and PAL by performing the TN/ATN comparison. The TN is in S, and the ATN from the previously executed test is in T. The corresponding bits in each register are compared at the scan-out-bus OR. If either bit is a 1, the inverted output of the OR is sensed as a 0.

For the result comparison, the mask is brought to S and compared with the value scanned into T during execution of the test. In hardcore tests, the mask is all 1's. This condition forces a pass or fail condition regardless of the value scanned into T during execution. A 0 output is sensed since the mask is all 1's. The expected result bit sets the 'pass' trigger and resets the 'fail' trigger. Using the mask in this manner, in conjunction with the MCW, causes the CE to decide whether to take the next test, an alternate test, or to terminate testing.

The significant bits of the MCW used during hardcore testing are 5, 6, and 7. MCW(5) is the UT bit, and, if set, causes the CE to stop after the test, regardless of the outcome of the test. MCW(6) is the CT bit, and, if set, causes the CE to stop if the test fails or to take the next test if the test passes. MCW(7) is the ERSLT bit which specifies whether a 0 or a 1 should be sensed at PAL following the result-comparison portion of the test. The combinations of the ERSLT and the output at PAL that determine the setting of the 'pass' or 'fail' trigger are:

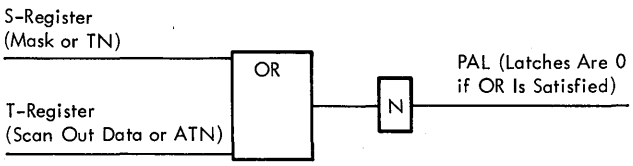
ERSLT Bit	PAL = 0	Set 'Pass' Tgr	Set 'Fail' Tgr
0	Yes	–	Yes
0	No	Yes	–
1	Yes	Yes	
1	No	–	Yes

The setting of the 'pass' or 'fail' trigger is then compared with the setting of MCW(5, 6) to determine the next operation, as follows:

MCW(5) (UT)	MCW(6) (CT)	Trigger Output 'Pass' Tgr	Trigger Output 'Fail' Tgr	Action
0	0	1	0	Continue – alternate test.
0	1	1	0	Continue – next test.
0	0	0	1	Continue – next test.
0	1	0	1	Stop – gate alternate test on restart.
1	0	1	0	Stop – gate alternate test on restart.
1	0	0	1	Stop – gate alternate test on restart.

Whenever the CE stops during hardcore tests (whether an error stop or an unconditional stop), the ATN is left in T. To restart the test, the tape must be backspaced the required number of times, after which depressing LOAD will IPL the tape from that point. As each test is brought in, the CE brings the TN of each sequential test into S. S and T are then compared, and, when PAL equals 0, the CE performs the test that is in the buffer at that time.

One of the preliminary steps in preparing the CE to perform ROS tests or FLTs is to set S and T to all 1's by setting the DATA switches to the DOWN position and depressing STORE (LADS A6503). This step is required because it is not known initially that these registers are functioning. By forcing 1's into all positions, the CE is forced to take the first test, which is test 1 on record 3. The rationale is that a failing bit in one register is compensated for by the bit in the other register, so that when S and T are compared, the result is 0 and the test is taken. Shown below is a single position of the scan-out bus; a similar position exists for each bit in S and T.



Note that a 1 input to either side of the OR satisfies the OR condition, and the inverted output is sensed as a 0. If neither input is a 1, the OR is not satisfied, and the inverted output is sensed as a 1.

Theory of Hardcore Tests

The hardcore tests were written for the 7201-02 to meet two objectives. The first is to test as much FLT (or ROS test) control hardware as possible to ensure that the results of the tests are meaningful. The second objective is to provide as much assistance as possible during repair of hardcore failures.

Repair procedures are contained in the 7201-02 CE maintenance manual and on LADS page A6511. Background information and basic hardcore theory follow.

An attempt is made to use only as much hardware as has been proven to work by previous hardcore tests. The first test determines whether it is possible to stop. The UT bit should make testing stop, but CT and Fail are also included to be redundant. (All error stops in hardcore stop in this manner.) This first test also proves that tests can be run. If they can't, no MCW 0–7 bits will come on; the tape may or may not run away.

In order to test the S- and T-registers, which are vital to FLT and ROS testing, searching is used. Remember that a TN "equals" an ATN when all 32 bit positions have 1 bits from either or both registers. To test for bits stuck on in S, the T-register (ATN) is set to 0 and a search is started. Searching should not stop until a TN with all 1's is found. Meanwhile, 32 tests are presented, each lacking only a single bit in the TN. If bit 6 is stuck on, the TN lacking bit 6 will seem to have all 1's, searching will stop, and this error stop test will be run.

Unfortunately, if the S-register is not failing, but bit 6 (or bit 38) is stuck on in the T-register, the same symptom will result. If comparisons cannot be made on bit 6 (PAL 6 stuck off), this same symptom results. Maintenance personnel must determine which failure occurred. The alternate method of testing for this failure is to let every error-stop TN be 0 while the T-register has been loaded with all bits except one by dummy tests which appear before each error stop. The data patterns are the same as before but are exchanged between S and T. Every other test is run, loading the T-register with its ATN and then searching across the following test.

Depressing LOAD should initiate an ATN search; passing from one test record to the next involves a restart IPL which should also initiate an ATN search. To test for this, the last test that is run in a record sets the T-register to search across the first test in the next record, which would be an error stop.

'Conditional terminate' should force the next test to run and inhibit searching. To test for this, the ATN of a test that passes (with the CT bit on) is set to search for an error stop. Without CT, 'fail' should force the next test to run, with testing proceeding as described above.

This covers most of the hardcore tests. These tests are designed to provide maximum isolation for all the basic failures which are latches and triggers stuck on or off.

Summary of Hardcore Tests

The hardcore tests have checked the CE for the following:

1. Ability to sense 1's in S and T as 0's at PAL.
2. Ability to sense 0's in S and T as 1's at PAL.
3. Ability to take next test.
4. Ability to perform a TN search.
5. Ability to stop on a failing test.
6. Ability to stop on a UT signal.
7. Ability to make a result-comparison and decide on next step.
8. That all data paths connected with the above functions are operating properly.

ROS Bit Tests

- ROS tests, two doublewords, are read from tape one at a time, alternately, into buffers 1 and 2.
- Test sequencing is controlled by ROS test sequencer.
- A test consists of: (1) fetching MCW from storage, (2) advancing CE clock, (3) comparing state of one ROSDR trigger with predetermined result.
- If 'pass' trigger is set, operation proceeds to next test; if 'fail' trigger is set, testing is terminated.

The remaining records on the ROS test tape contain the test patterns used to check the ROS bit planes. Each ROS test pattern consists of two doublewords in the same configuration as the hardcore tests. (See "Hardcore Test Highlights".) These tests are read from the tape one at a time, alternately, into buffers 1 and 2. At the completion of each read-in cycle, a 'TIC' pulse is generated (by the IOCE) which initiates the ROS test sequence.

Test sequencing is controlled by the ROS test sequencer. At the start of a test, the ROS test sequencer is set to maximum (7). Then, as each portion of the test is executed, it is decremented by 1. For each count of the sequencer, the CE is said to be in a certain "ROS state"; for example, if the ROS test sequencer equals 5, the CE is in ROS test state 5. When the count reaches 1, the test is complete and the CE waits until another 'TIC' pulse is received.

A test consists of fetching the MCW from storage, permitting the CE clock to advance a given number of cycles, and comparing the state of one of the ROSDR triggers with a predetermined result. If the state of the trigger matches its predicted value, the 'pass' trigger is set and the operation proceeds to the next test. If the actual and predicted values disagree, the 'fail' trigger is set and testing is terminated.

When the 'TIC' pulse is received, a TN comparison takes place. If the comparison is successful, the operation proceeds to scan in the MCW; otherwise, the operation waits. CE clock signals are distributed as long as the FLT counter does not equal 0. When the FLT counter equals 0, the CE clock is stopped and the expected result comparison is started. The address sequencer governs the loading of a portion of the ROSDR word (32 bits) into T. The mask is in S (the other half of the word that was loaded into the MCW), and the result comparison takes place. Until a 'TIC' or 'gap' pulse is not received, the operation returns to scan-in and repeats the test.

For the following discussion of the ROS test, refer to Diagram 6-115, FEMDM.

ROS Test State 7. A ROS test is started when the ROS test sequencer is set to maximum and 'Start ROS test' is set. This action occurs because one of the following conditions is present:

1. A 'gap' pulse is received from the channel, indicating an end of record. This condition is tested for during ROS test state 6.
2. The TN comparison failed during ROS test state 4; therefore, the next test is brought in.
3. Load was depressed. During ROS test state 7, the only operation is the decrementing of ROS test sequencer and inverting of buffer trigger.

ROS Test State 6. During ROS test state 6, the TN/ATN address is forced to storage controls, and a scan storage request for the TN/ATN word is initiated. The operation then waits for a 'TIC' pulse before progressing. When this pulse is received, ensuring that a test is in storage, the ROS test sequencer is decremented by 1.

ROS Test State 5. During ROS test state 5, the TN of the incoming word is gated to S, and the 'TIC' and 'gap' triggers are reset. The ROS test sequencer is then decremented by 1.

ROS Test State 4. During ROS test state 4, a 'scan out S and T' signal determines whether this is the test to be executed. This determination is accomplished by comparing the contents of S and T, via the negative-OR inputs to PAL (scan-out bus). S contains the TN of the current test obtained from the ROS test just read into storage during ROS test state 5. For the first test (hardcore test in record 3), T contains all 1's loaded from the DATA switches. For all subsequent tests, T contains the ATN of the previous test. This number should be the complement of the current TN. Thus, a successful comparison results in all 0's being sent to PAL.

PAL is then checked for an all 0 result. If PAL does not contain all 0's, indicating that the test currently in storage is not the one searched for, the ROS test sequencer is again set to maximum and decremented by 1. This action causes the CE to be again in ROS test state 6 and to wait for the next sequential test on the tape. This operation continues until either the entire tape has been searched or the correct TN has been found. If a valid TN cannot be found, this condition is known as a tape runaway.

If PAL does contain all 0's, indicating a successful comparison, the 'pass' and 'fail' triggers are reset and the ROS test sequencer is decremented by 1.

ROS Test State 3. During ROS test state 3, a storage request is made for the mask/MCW word, and the ROS test sequencer is decremented by 1.

ROS Test State 2. Highlights:

1. MCW fetched from storage is transferred to ST and subsequently distributed to address sequencer (bits 21–25), MCW register (bits 0–7, 20), ROSAR (bits 8–19), FLT counter (bits 26–31).
2. CE clock signals cycle ROS until FLT counter is reduced to 0 (one cycle for ROS tests).
3. When FLT counter equals 0, CE clock is stopped and result in ROSDR is scanned out to T.
4. ROS test sequencer is decremented by 1.

During ROS test state 2, the doubleword fetched from storage is transferred to ST, and the address sequencer, the FLT counter, and the FLT clock are reset. T now contains the MCW which is subsequently distributed as follows:

1. T(32–39, 52), which contains the ROS plane number [MCW(0–3)], the UT bit [MCW(5)], the CT bit [MCW(6)], the ERSLT bit [MCW(7)] is transferred to the MCW register. [MCW(20)] is used to indicate intentional stops. MCW(4) is not used and therefore contains 0. The ROS plane number is not used for the test but is displayed, in case of a failure, as a guide for maintenance personnel. For most ROS tests, except the hardcore tests, the UT bit is 0 and the CT bit is 1. The ERSLT bit is a 1 or a 0, depending upon the design of the ROS plane being tested.
2. T(40–51), which contains the ROS address of the plane to be tested, is set into ROSAR.
3. T(53–57), which contains the scan word address of the ROSDR bit to be tested, is transferred to the address sequencer.
4. T(58–63), which contains a count of the number of clock cycles needed to read out one ROS word, is transferred to the FLT counter and FLT clock.

At the same time, the 'scan counter control' trigger is set. The trigger output deactivates the ROS sense latch reset (until this time, the ROS sense latches have been held reset; therefore, no microprogram operations have been taking place) and sets the 'scan counter control' latch. This latch causes the ROS test sequencer output to be blocked, thus taking the operation out of ROS test sequencer control, and inhibits stepping the ROS test sequencer.

At this time, the bit test begins ROS functions using the address in ROSAR. The CE clock cycles ROS until the FLT counter (loaded from the MCW word and decremented by 1 in synchronism with the CE clock cycles) is reduced to 0. When the FLT counter equals 0, the 'cycle counter equals zero' latch is set and the 'MMSC' trigger is set to stop the CE clock and the test. The scan-out bus is now used to scan out the results. The 'SOROS' and 'sync' triggers are set, and the 'sync' latch is set to complete synchronization of the controls with the FLT clock. The ROS sense latches are reset, and register ingating is inhibited. The 'MMSC' trigger is then reset. The output of the scan-out bus is transferred to PAL and subsequently to T. The 'scan counter control' latch is reset, thus activating the 'reset ROS sense latch' signal and restoring control to ROS test state 2. A scan machine reset resets the latches and triggers used to control scan-out. The ROS test sequencer is decremented by 1.

ROS Test State 1. During ROS test state 1, the results in T and the mask in S are negative-ORed into PAL by the 'scan out S and T' signal. A scan storage request for the TN/ATN word is initiated and, depending on the ERSLT bit value and the PAL zero-result check, the 'pass' or 'fail' trigger is set as follows:

PAL	ERSLT Bit	'Pass' Tgr	'Fail' Tgr
= 0	1	Set	–
≠0	0	Set	–
= 0	0	–	Set
≠0	1	–	Set

If a 'TIC' signal has not arrived to indicate that the alternate buffer is full, the operation returns to ROS test state 3 to repeat the test. If an error or a 'TIC' or 'gap' signal occurred, the ROS test sequencer is decremented by 1, thereby going to ROS test state 0.

ROS Test State 0. During ROS test state 0, the word fetched in ROS test state 1 is loaded into ST, and a stop or continue decision is made. If testing is to be continued an ATN is specified and the ROS test sequencer is set to 7 and decremented by 1. If the test is to be stopped, the 'pass pulse' trigger is reset to stop the FLT and scan clocks and a 'reset FLT IPL' signal stops the tape at the end of the current record. The ROS test sequencer is set to 7, and the scan controls stop. To continue testing ROS, depress BACKSPACE FLT twice and LOAD.

When the ROS test stops because of a failure, the following information is displayed in the CE control panel indicators:

1. Failing plane (first digit of ROS address), in hex [roller 5, position 2, bits 0–3 (CE TEST ADDRESS)].
2. Failing ROS address (last two digits of ROS address), in hex [roller 1, position 3, S(0–7)].
3. Failing bit in decimal [roller 1, position 3, S(8–15)].

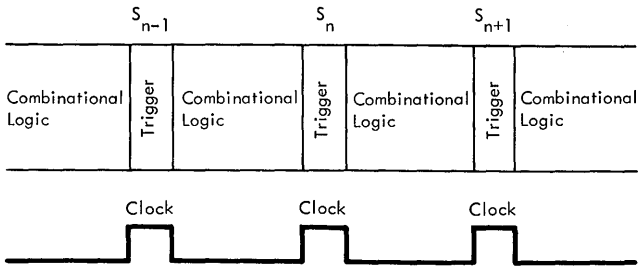
FLT TESTS

Introduction

- FLTs check CE logic at block level.
- 'Exit' trigger determines whether test passed or failed.
- FLTs have three basic phases: scan in, clock advance, scan out.
- Three categories of FLTs are hardcore tests, zero-cycle tests, and one-cycle tests.

FLTs are a unique maintenance concept in that CE logic is checked without executing CE instructions (i.e., without executing a program in the ordinary sense). In an FLT, fault detection is performed at the logic level. That is, FLTs are concerned with the logical function (OR, AND, INVERT) of a block rather than with its operational function (e.g., as an adder, counter, control) in the CE.

The FLT generator program treats the CE as a set of triggers which can take on a new state (S) every time a clock signal is generated:



Thus, FLTs are generated for groups of logic blocks. A group consists of an 'exit' trigger (a temporary storage element taking on new data at clock time and holding the information during not-clock time) and associated logic that can affect the trigger during an advance of a predetermined number of cycles.

The FLT generating programs analyze the group of logic blocks and determine, from a set of input values, an expected response at the output of the 'exit' trigger. (Note that although the 'exit' trigger determines whether the test passed or failed it is not being tested per se; the FLT tests the logic that precedes it.) Using the information gained by the analysis, the program generates a test pattern which it places on tape. Each test pattern is an FLT. At the same time, the program generates the documentation necessary to troubleshoot a failure detected by the FLT.

The application of an FLT can be broken into three phases:

1. Scan-in. Triggers are set to the desired state by a scan-in microprogram. Triggers that cannot be set are assumed to be at a predetermined value.
2. Clock advance. The number of clock signals specified by the FLT MCW are allowed to be generated. These signals advance the CE from one trigger level to the next. The 'exit' trigger assumes a value that is a function of the state of the CE during the previous cycle.
3. Scan-out. The new value of the 'exit' trigger is compared with the expected result. If the machine is operating properly, the two values should agree. If the new value and the expected result differ, a fault has been located and testing terminates† with reference to the FLT documentation.

FLTs can be divided into three categories:

1. Hardcore tests. Check the scan and normal CE logic necessary to run FLTs.
2. Zero-cycle tests. Determine whether a trigger value can be changed by scan in and also whether the new value can be sensed. Zero-cycle tests establish the machine capability to scan-in and scan-out before running one-cycle FLTs.
3. One-cycle tests. During these FLTs, data is scanned into the CE, the clock is allowed to run, and the 'exit' trigger is scanned out and compared with a known value. One-cycle tests check combinational†† logic within the CE.

Note that the terms "zero-cycle" and "one-cycle" do not refer to the number of clock cycles allowed after scan-in. For example, if, during a zero-cycle test, the 'exit' trigger requires a clock signal to set it, the clock must run for one cycle. These terms refer to test techniques rather than to any time element.

FLT Tapes

FLTs are stored on magnetic tape in the following order: hardcore tests, zero-cycle tests, one-cycle tests. Each FLT tape consists of thousands of tests, each concerned with a single sensitive path. Also contained on the tapes are tests preliminary to FLTs; i.e., designed to ensure that the CE is capable of performing FLTs and that the triggers to be tested can be set or reset and the change sensed.

†Termination is conditional upon the contents of the MCW.

††Combinational logic is all the logic required to pass the state of one trigger to the next by executing a specified number of clock cycles.

Each tape is divided into three sections: hardcore tests, zero-cycle tests, and one-cycle tests. The one-cycle tests are considered the true FLTs because these test combinational logic. A brief description of the three kinds of tests on the FLT tape follows:

1. Hardcore tests. The test tape begins with hardcore tests to check out the CE hardware necessary for operating the FLTs. Hardcore tests determine that the S- and T-registers are functioning properly, that their bit content can be correctly sensed at PAL, and that the CE can make decisions based on the outcome of a test and then act on that decision.
2. Zero-cycle tests. Next in the testing sequence are the zero-cycle tests. These tests verify that the 'exit' trigger status can be changed and that the change can then be observed or sensed. Zero-cycle tests set the trigger, using either a special scan input or a normal machine path, and then verify the change in status of the trigger being tested. Because clock signals are needed to set most triggers, the CE clock is allowed to cycle just enough to set the trigger. Upon completion of the zero-cycle tests the CE's ability to run FLTs has been verified.
3. One-cycle tests. These tests make up the bulk of the FLT tape. They vary in the amount of logic checked and run in sequence until a failing test is encountered, whereupon testing is terminated and the failing test number is displayed in the S-register. One-cycle tests require two or three clock cycles to set a trigger.

Tape Generation

- FLT tape is computer-generated from ALD data.
- Program develops sensitive tree with entry points and terminating in an 'exit' trigger.
- Tests are printed out in two formats: test, including entry pattern, and listing of sensitive points (SCOPEX) which is used in troubleshooting.
- New tests are added to existing tape, using FLUT.

The FLT tape is computer-generated from ALD data. Using this data, a special program develops tests that, when executed, affect triggers indicatable on the CE control panel (designated 'exit' triggers). The program works back from all entries into each of these triggers, searching the logic path for "sensitive" points; i.e., points at which an error or a failure would propagate to the 'exit' trigger. Only one fault is assumed for each sensitive point.

The logic that feeds these sensitive points makes up a sensitive net, and, as the number of nets grows, the combinational logic involved resembles a tree; the 'exit' trigger forms the tip of the tree, and the sensitive nets make up the body of the tree. The search continues until the program encounters another trigger that can be used as an entry point into the sensitive tree. Usually, several entry points into a particular tree are available. However, entry points are selected on the basis that data injected at the entry points will propagate through to the 'exit' trigger.

The program also selects micro-orders that will move data from the entry points to the 'exit' trigger. Upon completion of the search, the program has developed a sensitive tree with entry points and terminating in an 'exit' trigger. The program also has developed an entry pattern that results in a predictable status of the sensitive nets and a predictable change at the 'exit' trigger. Once this pattern has been developed, an evaluator program verifies its correctness.

Upon verification, the test is printed out in two formats: the test itself, including the entry pattern, and a listing of the sensitive points within the tree. The first of these becomes the taped FLT; the latter is the scoping documentation (scoping index, or SCOPEX), used in troubleshooting a failing test.

The entry pattern is scanned into the CE, micro-instructions are selected to move the data through the sensitive tree during a given number of clock cycles, and the 'exit' trigger is then observed (scanned out) to determine whether it is at the predicted value for the test. If it is not at the predicted value, the test has failed, and maintenance personnel may then repeat the failing test continuously and scope the sensitive points on the tree to find the net with the failure in it. Certain sensitive points appear in more than one test, and the same 'exit' trigger may be the observation point for more than a single sensitive path. Thus, newly tested points are indicated as being newly tested on SCOPEX to indicate to maintenance personnel that this is the first time these sensitive nets have been encountered and tested.

By verifying that ANDs, ORs, and INVERTs are functioning as designed, FLTs verify the operating capability of the CE, although specific computer functions are not performed. FLTs ascertain that the CE is operating according to design specifications and therefore should be capable of functioning as a CE.

After the FLT has been evaluated as a valid test, it becomes part of the FLT tape and is distributed for field use. New FLTs are continually being generated, some to test new logic that results from engineering changes and others to test areas of the CE that are not now being checked. New tests are incorporated into existing tapes with the Fault Locating Utility (FLUT) program. Normally, however, a complete tape is sent to the field incorporating both old and new tests.

FLT Hardcore Tests

The first few records on the FLT tape contain tests that ensure that the hardcore logic necessary to run zero-cycle and one-cycle tests is operational. Hardcore logic is defined as all the logic, scan and normal, necessary to load the FLTs into storage, to scan-in, to scan-out, to make decisions based on the outcome of a test, and to act on those decisions.

CE hardcore hardware testing is subject to the following limitations:

1. Scan hardware is not tested directly; however, it is exercised in the hardcore portion with tests designed to isolate the trouble.
2. The SCI is not tested in the FLTs, but it must be functioning properly for FLTs to be run. Also, main storage and storage buses must be operating. Main storage may be checked by means of ripple tests.
3. The channel and tape drive (or disk) used to read in the FLTs are not tested. These units must be tested by means of manual controls on the IOCE.
4. Local storage is not needed to run FLTs and, therefore, is not tested. Local storage is checked by means of ripple tests.
5. The ROS microprogram and ROS must be fault-free to run FLTs. Therefore, because hardcore tests do not check ROS, ROS tests should be run before FLTs.

Zero-Cycle Tests

To further check the operation of the scan hardware, zero-cycle tests determine whether the scan-in and/or scan-out paths are operative. Zero-cycle tests check only those triggers displayed on the CE control panel. In these tests, a pattern is scanned into the machine, the clock is not advanced, and the 'exit' trigger is observed. If the trigger has a scan-in path, three tests are performed: one for the reset state, one for the set state, and one for the reset state again. If the trigger has no scan-in path, only one test is performed for the reset state. While one trigger is being tested, other triggers can assume various states. Whenever possible, random states are used to simulate the combinations that may be used in a normal test and to reveal interaction between triggers.

The functions checked with zero-cycle tests include:

1. Reset to triggers (this is, in effect, a scan-in-zero operation).
2. Scan-in path to triggers.
3. Ability of a trigger to hold its value in the absence of a clock signal.
4. That scan-out bus gating signals can be generated.

One-Cycle Tests

One-cycle tests are the true FLTs. The input, produced by the FLT generating system, is a test that detects and locates at least one fault. The input pattern is scanned into selected triggers, and the CE is allowed to advance a given number of cycles. The result, which is in the 'exit' trigger, is compared with the value expected for a correctly operating machine.

If the value in the trigger does not agree with the expected result, testing is terminated and the failing TN is displayed in S(0–15) for reference to the SCOPEX.

The SCOPEX is a series of lists, one for each test. Each list is headed by the test number in hex, followed by a row of asterisks, and consists of several lines each line referring to a pin in the machine. If a pin in the machine is contained in a list, the net which feeds that pin is sensitive for the test pattern applied; a failure on the card, which can be observed with an oscilloscope at that pin, would cause the test to fail.

Each FLT on the test tape consists of 17 doublewords, numbered 0 through 16. This is one complete test. Each test occupies one of two areas in storage. Buffer 1 begins at PSA location 100 (hex) and contains 17 doublewords. Buffer 2 begins at 17 PSA location 200 (hex) and also contains 17 doublewords.

While one test is being executed from one buffer, the other buffer is being filled from the IOCE. Filling the buffers is a sequential process, and the processing of the FLTs is in the same sequence, without further addressing.

The contents of the 17 doublewords are as follows:

1. Doublewords 0 and 2 through 15 contain bit patterns that are scanned into the CE.
2. Doubleword 1 contains a mask and the FLT MCW. The mask defines the 'exit' trigger; the MCW, in the right half, contains control information about the test.
3. Doubleword 16 contains the TN in both true and complement form and the ATN in the same format. Doubleword 16 is not read into the CE during test sequencing unless a fault is encountered or an ATN search is performed.

At the satisfactory completion of a test, the 'pass' trigger is set, the address sequencer is set to 15, and word 15 of the test is the first word scanned in again. When a test fails, the 'fail' trigger is set. After a 'TIC' pulse is received from the IOCE, the last thing the CE does is to fetch word 16 from storage and leave it in the S- and T-registers. If the maintenance personnel desires to skip this test and continue with the remainder of the tests on the tape, he may depress the BACKSPACE FLT pushbutton twice, then LOAD. The

signals generated by depressing these pushbuttons causes the I/O device to backspace to the beginning of the record and to start reading the tests into storage again. As each buffer is filled, the TN in the left half of word 16 is placed in the S-register, and a comparison is made between the new TN in the S-register and the ATN in the T-register by using the scan-out S and T facility. The result is 0 for the test immediately following that test in which a failure occurred. Upon detection of this 0 result in PAL, the CE again begins testing and continues until the next failure. This procedure is the only way of getting past a failing test and continuing the test tape sequence.

Operational Analysis

FLT Tests

- Each FLT checks small portion of logic by setting up conditions that affect 'exit' trigger which can then be sensed for proper output.
- If error occurs, test is terminated and SCOPEX is used with failing TN identification to determine test points to scope.
- FLT tapes consist of hardcore, zero-cycle, and one-cycle tests.
- FLT test consists of 17 doublewords.

As discussed earlier, each FLT checks a small portion of the logic by setting up certain conditions that affect a specific trigger (known as an 'exit' trigger) which can then be sensed for the proper output. Therefore, each test on the tape sets up the CE to the proper status (certain triggers set) that results in a particular output of the logic under test. After the CE has been set up for a test (scan-in), the CE clock is advanced a sufficient number of cycles to change the status of the 'exit' trigger. The indicator associated with this trigger is then selected (scan-out) and compared with an ERSLT bit in the FLT MCW, set as part of the scan-in routine. If the values are equal, the test passes; if unequal, the test fails. Because each test is repeated a number of times, both the PASS and FAIL indicators may be on, which is interpreted as an intermittent failure.

A summary of an FLT includes:

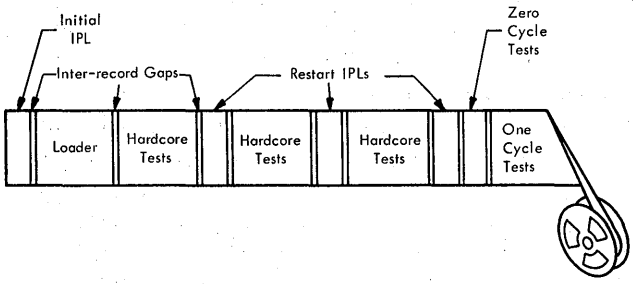
1. Load test into storage.
2. Scan into CE triggers.
3. Advance CE clock.
4. Stop CE clock.
5. Scan-out to T (scans desired indicator to T).
6. Load mask into S.
7. Scan-out S and T (compares 'exit' trigger with expected result and sets 'pass' or 'fail' accordingly).
8. Repeat steps 2–7 until a 'TIC' or 'gap' is sensed.
9. Go to next test or terminate and display failing TN.

The action following this sequence (assuming an error or fault) requires locating the SCOPEX page with the failing TN identification, scoping the test points listed, and replacing the failing card(s).

FLTs are rarely run singly. So little time is involved that it is easier to run the entire series on a tape or until a failing test is encountered. The failing test may then be repeated until the fault is isolated.

FLT Tape

The makeup of the FLT tape is similar to that of the ROS test tape and contains the following records:



1. Record 1, Initial IPL – Contains the 24-byte "bootstrap" program necessary for an IPL operation. When LOAD is depressed, the three doublewords of this record are read into storage locations, as follows:

Storage Location (Hex)	Word	Contents
0		Loader ID
8	CCW 1	Read command to read 8 bytes to location 0, and chain command to location 10.
10	CCW 2	Read command, 28 (hex) bytes into 18 (hex). Chain command to location 18.

2. Record 2, Loader – Contains the loader program that reads in the FLTs. The IPL program in record 1 reads this record into storage locations, as follows:

Storage Location (Hex)	Word	Contents
18	CCW 1	Read command. Read 8 bytes into location 0 and chain data.
20	CCW 2	Read command to read 88 (hex) bytes to buffer 1 (location 100), and a chain data tag to location 28.

28	CCW 3	TIC command to location 30.
30	CCW 4	Read command to read 88 (hex) bytes to buffer 2 (location 200), and a chain data tag to location 38.
38	CCW 5	TIC command to location 20.

3. Record 3, Hardcore Tests 1 and 2 – Contains the first FLT hardcore test. The hardcore tests for FLTs are identical with the ROS test hardcore tests except for their format. The FLT hardcore tests have the following format:

Storage Location (Hex)	FLT Word	Left Word	Right Word
100	0	0	0
108	1	Mask	MCW
110	2	0	0
118	3	0	0
120	4	0	0
128	5	0	0
130	6	0	0
138	7	0	0
140	8	0	0
148	9	0	0
150	10	0	0
158	11	0	0
160	12	0	0
168	13	0	0
170	14	0	0
178	15	0	0
180	16	TN	ATN

Note that all locations except those containing FLT words 1 and 16 are blank. The blank locations are not used in hardcore tests.

4. Record 4, IPL 2 – The first hardcore test causes a stop. When LOAD is depressed, the following IPL program replaces the IPL 1 program:

Storage Location (Hex)	Word	Contents
0		Loader ID
8	CCW 1	No-op chain command.
10	CCW 2	No-op chain command.

5. Records 5–8, Hardcore Tests – Contain the remaining hardcore tests.
6. Zero-Cycle Tests – The next series of records on the FLT tape are the zero-cycle tests. These tests further check the scan logic necessary to run the main FLTs (one-cycle tests). They determine whether the scan-in and scan-out paths are operative. Scan-in and scan-out tests can exist only for triggers which have a scan-out path. In these tests, a pattern is scanned into the machine, the CE clock is allowed to advance, and the output trigger is observed. If the trigger has a scan-in path, three tests are performed: for a reset state, for a set state, and again for a reset state. If the trigger has no scan-in path, only one test is performed, and this test ascertains that the trigger is at its reset state. While one trigger is being tested, other triggers can take on various states. Whenever possible, random states simulate the combinations which may be used in a one-cycle test and reveal interaction between triggers.
7. One-Cycle Tests – The remainder of the tape contains the one-cycle tests which are the true FLTs. The input has been generated by the FLT generating system and found to be a test that will detect and locate at least a single fault. An input pattern is scanned into the predetermined triggers, and the CE clock is allowed to advance; the result, which is in the 'exit' trigger, is compared with the value expected for a machine operating correctly.

The last test in each record is a dummy test, which keeps the IOCE from reading across the interrecord gap before the result of the last one-cycle test is known. Thus, if the last one-cycle test fails, a stop-scan sequence can be executed before the interrecord gap is reached.

FLT Format: Zero-cycle and one-cycle FLTs have the same format (Table 4-2), consisting of 17 doublewords. The doublewords (scan-in words) are used as follows:

Table 4-2. FLT Format

Storage Location (Hex)	Scan-In Word*	Left Word	Right Word
100 and 200	0	S	T
108 and 208	1	Mask	MCW
110 and 210	2	Q	Q
118 and 218	3	A	Misc Triggers
120 and 220	4	B	Misc Triggers
128 and 228	5	D	Misc Triggers
130 and 230	6	PSW	IC
138 and 238	7	E, R	Misc Triggers
140 and 240	8	K	PIR
148 and 248	9	Select	IOCE MCK INT
150 and 250	10	External	(Spare)
158 and 258	11	Dar Mask	Dar Mask Parity
160 and 260	12	(Spare)	DAR
168 and 268	13	L	M
170 and 270	14	X	Y
178 and 278	15	N	(Spare)
180 and 280	16	TN	ATN

*Refer to LADS A6611 and A6641 for bit assignments.

1. Scan-In Words 0 and 2–15. Contain the scan-in test pattern, which is distributed to registers and triggers as shown in Table 4-2. The data establishes a CE state before performing the FLT.

2. Scan-In Word 1, Left Half (Mask). Specifies the 'exit' trigger to be sensed. This is accomplished in the scan-out bus by ORing the scan-out word containing the 'exit' trigger status with the mask bits. The mask contains all 1's except for a 0 in the position corresponding to the trigger being sensed. The output of the OR is inverted so that if the 'exit' trigger is set, the value sent to PAL is 0.
3. Scan-In Word 1, Right Half (MCW). Contains the control information necessary to run the test:

MCW Bits	Scan-In Word 1 Position	Contents
0–3	32–35	(Used for display only)
4	36	LW bit
5	37	UT bit
6	38	CT bit
7	39	ERSLT bit
8–19	40–51	ROS address (bits 73–84 of input pattern)
20	52	(Used for display only)
21–25	53–57	Scan-out address
26–31	58–63	Cycle count

4. Scan-In Word 16. Contains the TN and ATN as follows: bits 0–15, TN; bits 16–31, 1's complement of TN; bits 32–47, is complement of ATN; bits 48–63, ATN.

FLT Test Setup

Several controls on the system control panel must be operated to initiate FLTs. The procedure for running FLTs appears in LADS A6503. However, a short description of the setup is included here because it affects the operation.

Diagram 6-116, Sheet 2, FEMDM, shows the start of an FLT. The FLT tape is mounted first. SYSTEM RESET is now depressed to place the CE in stop loop with the 'manual' trigger set. The LOAD UNIT switches are set to the address of the tape unit holding the test tape, the TEST MODE, ROS/PROC/FLT switch is set to FLT, and the CE CHECK switch is set to DSBL. Going into FLT test mode causes the 'FLT test' trigger to be set. The DATA switches are set to all 1's, and STORE is depressed. This action places all 1's into ST so that T now contains an ATN for the first hardcore test. Lastly, LOAD is depressed, initiating a normal IPL operation which reads in IPL 1.

IPL 1. Depressing LOAD with the CE in FLT test mode initiates operations that are identical with a normal IPL; i.e., 24 (dec) bytes are read from the selected device into the first three doubleword locations of main storage under ROS microprogram and IOCE control. At this point, the CE is idle, waiting for a 'TIC' signal from the IOCE. Meanwhile, the IOCE IPL operation reads in record 1 of the FLT tape and executes the program specified by record 1. (See "FLT Tape".) As a result, 40 bytes (record 2) are read into storage, starting at location 18 (hex). Record 2 contains the loader program that will read each FLT into the proper buffer area in storage.

After record 2 has been read in, command chaining causes CCW2 in record 2 to be executed. This CCW is a Read command to read test record ID into location 0 which contains a read CCW, reads test into 100. When the IOCE has finished executing the TIC command, in 28, it sends a 'TIC' signal to the CE. As soon as this signal is received, the IPL microprogram is continued, and the CE proceeds as though performing a normal IPL. Then the FLT microprogram is initiated. ROS control now places the CE in scan mode. The CE is now ready to run the first hardcore tests.

Loader

- FLTs are read into two buffer areas (100 and 200, hex) in the PSA area.
- Each time a buffer is filled, 'TIC' pulse informs CE that test can be performed.
- Read-in operation continuously checks for errors.
- Test is repeated until 'TIC' pulse is received.

The loader IOCE program read in by IPL 1 is used to transfer all FLTs, whether hardcore, zero-cycle, or one-cycle tests, to main storage. The test data is read into two buffer areas in main storage [starting at locations 100 (hex), designated buffer 1, and 200 (hex), designated buffer 2]. Each time a buffer is filled with a single test, a 'TIC' pulse informs the CE of this fact and the test can be used by the CE.

The tests are read into the buffer areas on a sequential basis. For example, test 1 is read into buffer 1. As test 1 is being run, buffer 2 is being filled with test 2. As test 2 is being performed, buffer 1 is being refilled with test 3, etc.

Parity and tape errors are not retried automatically and manual intervention is required following a stop under these conditions. Channel-control checks may also occur, which immediately halt the test procedure.

Once a test is begun, it is repeated until a 'TIC' pulse is received, indicating the next test is ready for processing. No count is made of the number of times a test is run, as this is a function of the data rate of the I/O device involved.

During normal sequential processing of FLTs (no faults encountered), the address sequencer is set to 15 and the 16 doublewords composing a single FLT are scanned into the

CE. The following is a summary of the read-in and test sequence:

1. While the FLT is being scanned into the CE, a continuing check for storage and/or input errors is made. Storage errors might occur between storage and the CE. Input errors might occur between IOCE and storage while the next FLT is being read into the alternate buffer.
2. When scan-in is complete, the 'scan mode' trigger is reset, and an unconditional branch to the ROS word addressed by MCW(8–19) is performed.
3. The 'scan counter control' trigger is set, and the FLT counter keeps track of the number of CE cycles specified by MCW(26–31). The CE clock is stopped when the FLT counter equals 0.
4. Following a successful scan-in and clock advance, a scan-out operation places the test result in T. The combination of MCW(4) and MCW(21–25) determines the scan-out word that is transferred to T.
5. At this point, the mask is fetched from storage and transferred to S. A result comparison between the mask and the scanned-out word is made, and the 'pass' or 'fail' trigger is set, depending on zero reset and MCW(7). The next operation to be performed is determined by pass, fail, unconditional-, or conditional-terminate conditions.
6. Execution of this FLT is repeated until a 'TIC' or 'end of record' ('gap') signal is received. When either is received, the decision to stop or continue is made, and appropriate controls are set.

Hardcore Tests

Functionally, the hardcore tests at the beginning of the FLT tape are identical with the ROS hardcore tests. The differences are in the format and control of the tests. FLT hardcore tests have the same format as zero-cycle and one-cycle tests except that all scan-in words other than words 1 and 16 are 0's. Word 1 contains the mask and MCW, and word 16 contains the TN/ATN. These words are used in the same manner as in the ROS hardcore tests.

FLT hardcore tests are controlled by the FLT ROS microprogram instead of by hardware, as in the ROS hardcore tests. Successful testing terminates on three hardcore stops (correct stops), and, upon restart, the CE enters zero-cycle tests. Maintenance personnel may attempt isolation and repair on an error stop encountered during hardcore tests according to the procedure in LADS A6511.

Zero-Cycle and One-Cycle Tests

- Scan-in loads CE with test data.
- Test cycle allows CE to act on scan-in data.
- Scan-out collects data after clock advance.
- Mask is compared with scan-out data.
- Testing ends or continues, depending on result comparison, with expected result [MCW(7)].

Each FLT follows a similar five-step routine under ROS control (with the exception of scan-in and scan-out, which may be either under ROS or hardware control). The FLT sequence may be summarized as follows:

1. Scan-In. When a storage buffer has been filled with an FLT and the CE receives a 'TIC' pulse, scan-in begins. The address sequencer is set to 15 by a micro-order, and words 0 through 15 are read into the CE. These words enter the CE via Q or ST and are scanned into various registers and triggers throughout the CE, using normal data paths, to set up the machine environment for a particular test. During scan-in, certain micro-orders are interpreted for scan control rather than for functional operations. From T, the MCW is transferred to the address sequencer, the FLT counter, and the MCW register when the address sequencer equals 1. When the address sequencer equals 0, scan-in is completed except for ST, and the test-cycle phase is entered. A micro-order transfers T to ROSAR, and, on the next cycle, ST is scanned in.
2. Test Cycles. The count in the FLT counter, set by the MCW, determines the number of cycles taken by the CE. A micro-order, specified by MCW(8–19), controls this portion of the test. When the FLT counter equals 0, the CE clock is stopped and scan-out begins.
3. Scan-Out. The status of the 'exit' trigger is placed in T. The scan-out word containing the status of the 'exit' trigger is specified by the count in the address sequencer. MCW(4) determines whether the right or left word contains the 'exit' trigger status. If the count in the address sequencer is 14 or greater, scan-out is controlled by hardware. If the count is 13 or less, scan-out is controlled by ROS.
4. Result Comparison. The 'exit' trigger is compared with a known value, one that is predicted on the basis of the information scanned in. This comparison is accomplished by fetching the mask from storage, placing it in S, then ORing the scan-out word and the mask in the scan-out bus. The mask contains all 1's except for the position corresponding to the 'exit' trigger, which is a 0. Assuming the 'exit' trigger is set when it is ORed with the 0 in the mask, the output is transferred to PAL as a 0. Next, the CE compares the output of the OR with MCW(7), the ERSLT bit. With all 0's in PAL and MCW(7) = 1, the CE determines that the test has passed and sets the 'pass' trigger. If the comparison is unfavorable, the 'fail' trigger is set. Because the test is

continually repeated until the next 'TIC' pulse is received, both the 'pass' and 'fail' triggers can be set. After the result comparison, the CE must decide whether to terminate testing or continue.

5. Terminate or Continue. This decision is a major point in the FLT sequence. Four triggers determine what the CE will do next: 'pass', 'fail', 'UT' [MCW(5)], and 'CT' [MCW(6)]. The CT bit is always set in current FLTs (except for certain hardcore tests). Depending upon the setting of these triggers, the decision that will be made is as follows:

Trigger Output 'UT'	'CT'	'Pass'	'Fail'	Action
0	0	1	0	Continue – alternate test.
0	1	1	0	Continue – next test.
0	0	0	1	Continue – next test.
0	1	0	1	Stop – gate alternate test on restart.
1	0	1	0	Stop – gate alternate test on restart.
1	0	0	1	Stop – gate alternate test on restart.

The CE repeats the current test until the 'TIC' pulse is received from the channel. At that time, the decision to stop or continue is made. If the CE is to continue, the test in the opposite buffer is scanned in. No count is made of the number of times the test is repeated.

6. TN/ATN Comparison. This comparison is accomplished in hardcore and whenever the LOAD pushbutton is depressed. The TN located in word 16 is compared with the ATN left in T by the last test to be executed. Also, a specific TN may be entered into the T-register via the DATA switches and STORE pushbutton and the CE searches for this number to the exclusion of all other tests. As FLTs are now set up, the ATN in each test is the number of the next FLT on the tape (with the exception of hardcore tests).

Scan-In Highlights

- Test pattern establishes trigger status before test.
- Test words are read into S, T, or Q, then distributed throughout CE under microprogram control.

The scan-in portion of an FLT consists of fetching a test pattern to establish a trigger status before a test. Scan-in test words are read into S, T, or Q from the FLT buffers. From these registers, the data is transferred throughout the CE, under microprogram control, via special scan circuits and normal CE paths. Transfer paths are determined by the microprogram, which is repeated without variation for each FLT. At the end of scan-in, the CE clock is allowed to cycle the number of times required to condition the 'exit' trigger. Scan-in fetches 16 words from storage into the CE.

A scan-in is started with the address sequencer set to 15 (Diagram 6-116, Sheet 2, FEMDM). A scan storage request fetches scan-in word 15 of the record. This word is transferred to ST, and the address sequencer is reduced by 1.

Another scan storage request is made for word 14. Scan-in continues to operate for words 13–2 exactly as it did for word 14. Word 1 (mask and MCW) is loaded into ST, and a check is made for a 'TIC' or 'gap' pulse. Before word 0 is transferred into the CE, the mask and MCW (word 1) are transferred out of ST. Diagram 6-116, Sheet 4, shows that T(32–39) is transferred to the MCW, T(53–57) is transferred to the address sequencer, T(58–61) is transferred to the FLT counter and FLT clock, and the 'scan counter control' trigger is set. Word 0 is loaded into ST after T(40–51) is loaded into ROSAR. The 'scan counter control' latch is then set to allow the test to be performed.

Test Cycles. The CE clock is allowed to advance a number of cycles as specified by MCW(26–31). A portion of the microprogram, starting at the address specified by MCW(8–19), is performed. Note that at this point the CE seems to be running a program. However, this is not the case. The micro-orders being performed merely allow the scan-in test pattern to be transferred through the logic being tested to change the state of the 'exit' trigger. If setting these triggers at scattered points causes three legs of an AND to be tested, then 1's are directed to those triggers when testing the AND. If the three triggers happen to be those normally set during I-Fetch, for example, this fact is incidental to FLT testing.

As soon as the CE clock cycles begin, the FLT counter is stepped in synchronism with the clock. When the proper number of cycles have been taken, the FLT counter steps to 0, which causes the 'cycle counter equals zero' latch to be set. The 'SOROS' and 'MMSC' triggers and the 'sync' latch are then set. The 'MMSC' trigger stops the CE clock. As soon as the 'sync' latch is set and the operation is synchronized with the scan and FLT clocks, the 'MMSC' trigger is reset. The operation proceeds under ROS or scan logic control, depending on the value in the address sequencer.

Scan-Out. During scan-out (Diagram 6-116, Sheet 3), the 'exit' trigger status is transferred from the indicator driver circuits, through PAL, to T. This path is the same path used for logout operations.

The word scanned out is determined by the scan-out address in the address sequencer. The setting of the address sequencer determines which roller switch on the system control panel contains the desired information, and MCW(4) determines whether the right or left half of the word should be scanned out. Scan-out is first controlled by

scan hardware and then by the microprogram. The hardware-controlled portion of scan-out is always performed first regardless of the value in the address sequencer, although only those fields addressed by a value of 14 or greater are scanned out. If the 'exit' trigger is in a scan-out word whose address is 13 or less, the field is scanned out under microprogram control.

ROS control is resumed by forcing one address, if the address sequencer is greater than 13, and another address if it is less than 14. The microprogram started by the former is ready to perform a result comparison because the 'exit' trigger value is in T. The latter program first branches on MCW(4) to determine whether the left or the right half of the doubleword addressed by the address sequencer should be scanned out. The LSWR is gated to T by a normal CE gating signal; therefore, a branch must be made on address sequencer equal 13. Because the LSWR is contained in the right half of word 13, a branch on address sequencer equal 13 is unnecessary if MCW(4) equals 1. By means of these branches, the microprogram transfers the field to be tested to T and thus begins the result comparison.

Result Comparison. FLTs check logic by moving data through a group of logic to an 'exit' trigger and by causing a change in the state of this trigger. To determine the success or failure of a test, the change in the 'exit' trigger must be sensed on the basis of the data entered into the logic. This change is predictable, and the data on the test tape designates whether the trigger is to be a 1 or a 0 at the conclusion of each test. The expected state of the trigger is contained in the MCW as the ERSLT bit.

The scan-out S-and-T function (Diagram 6-116, Sheet 5) determines the setting of the 'exit' trigger by comparing a mask with the scan-out word containing the status of the trigger. Scan-out places the selected scan-out word in T. The mask word, fetched from storage, is placed in S. The mask contains all 1's except for the bit position corresponding to the 'exit' trigger in T. This bit position is a 0, and therefore the 'exit' trigger determines the setting of PAL.

If the 'exit' trigger is set, a 0 is transferred to the corresponding PAL bit; if reset, a 1 is transferred to PAL. Thus, PAL equals 0 only if the 'exit' trigger equals 1. If the ERSLT bit [MCW(7)] also equals 1 or if PAL does not equal 0 (indicating that the 'exit' trigger is reset) and the ERSLT bit equals 0, the 'pass' trigger is set. If MCW(7) equals 1 and PAL does not equal 0 or if MCW(7) equals 0 and PAL equals 0, the 'fail' trigger is set. In any case, if both the 'pass' and 'fail' triggers are set, the 'intermittent' trigger is also set.

Terminate or Continue. When the mask was loaded into S, the address sequencer was set to 16 and a scan storage request was initiated to fetch scan-in word 16 (TN/ATN). At the same time that the mask is compared with the scan-out word, scan-in word 16 arrives from storage and is gated to ST. The result in PAL and the ERSLT bit are used to set the 'pass' or 'fail' trigger and thus record the result. The address sequencer is then decremented, resulting in a repeat of the same test just executed.

Assuming the test was successful and no errors were found, the operation described is repeated until the buffer is filled with the next test or until an end-of-record gap is signalled. These conditions are repeatedly sampled for during scan-in. If a 'TIC' or 'gap' pulse is received, scan-in execution is abandoned. The address sequencer is set to 16, and the TN/ATN of the test in progress is fetched and loaded into ST. After a system reset which clears the CE registers, a stop or continue check is made (Diagram 6-116, Sheet 5). If a continue condition results, the operation proceeds to the setting of scan mode to start the next or alternate test as determined by the stop or continue controls. In case of a stop conclusion, the 'stop scan' signal causes the CE to drop 'IPL' to the IOCE and reset it to the halt loop with 'subsystem reset'. The UT bit [MCW(5)] is set, and the CE enters a microprogram loop.

After a stop condition, manual intervention is necessary. The operation then depends upon the pushbutton depressed, as follows:

1. START (with SCAN MODE, REPEAT switch off). Runs the test that causes the stop one time and causes another stop.
2. START [with SCAN MODE, REPEAT switch in the Repeat (down) position]. Continues to run the test that caused the stop until SCAN MODE, REPEAT is turned off.
3. STORE. May be used to set a different ATN into T. (BACKSPACE FLT and then LOAD are used.)
4. BACKSPACE FLT. Sends 'backspace' and 'IPL IOCE (X)' to the IOCE, which causes it to backspace one record. To restart after a failing test, BACKSPACE is depressed twice (backspace to the front of the test record, then to the front of the restart IPL record which precedes each test record), then LOAD.
5. LOAD. Initiates a subsystem reset and an IPL. Resets the MCW and 'buffer 1' and 'fail' triggers. Sets the 'pass' trigger. This combination causes an ATN search.

TN/ATN Comparison. Every FLT contains the TN and the ATN of the test which the CE is to execute upon completion of the current test. This information is contained in word 16 and consists of the TN and ATN in both true and complement form.

The hardcore tests verify the ability of the CE to conduct a TN search by comparing the TN (in S) with the ATN (in T), using the 'scan out S and T' signal. A 0 output indicates a favorable comparison, and the CE then executes the test. If the inverted-OR output is not sensed as all 0's,

the CE rejects the test and continues searching. The bit configuration in hardcore tests cannot strictly be called a TN, but the functions of comparison, acceptance, or rejection of the test are valid.

During zero-cycle and one-cycle tests, the TN search is rarely used. If the CE does not encounter a failing test once it has entered the FLT sequence, word 16 is not even used. The address sequencer is set to 15, and scan-in begins immediately.

However, if a failing test is encountered, the last action the CE takes is to place the ATN in T. The test immediately following the failing test has a TN that is the complement of the ATN. After manually restarting, when this TN is brought into S and compared with the ATN, a 0 output results, and the CE begins testing again with this test. This action allows the maintenance person to get past a failing test when he wishes to do so. This situation could happen, for example, when an FLT is failing because of an engineering change and the maintenance person is aware that failure is not due to a malfunction. Each FLT has the TN of the next sequential FLT as its ATN.

For example, assume the computer has just run TN 01 09 and the test has failed. The next test is TN 01 0A. BACKSPACE FLT is depressed twice, followed by LOAD, to get past the failing test. The TN of each test is the record is brought into S for comparison with the ATN left in T by the failing test. The comparison is favorable when the test immediately following the failing test is encountered, and the CE resumes testing at that point. TN 01 09 leaves the following configuration in T (right half of word 8):

ATN Complement	ATN
FE F5	01 0A

TN 01 0A, the next test, has the following configuration in the left half of word 8 which was brought into S:

TN	TN Complement
01 0A	FE F5

The following binary bit configuration is scanned out:

T: 1111 1110 1111 0101 0000 0001 0000 1010

S: 0000 0001 0000 1010 1111 1110 1111 0101

PAL: all 0's

The CE resumes testing with TN 01 0A and continues until another failure is encountered or to the end of the testing sequence.

RIPPLE TESTS

The ripple tests may be used to exercise either main storage or local storage and several functional units within the CE. The tests use only the internal hardware of the CE, including several ROS words, and storage. The procedure stores the contents of the DATA switches in all locations of storage or displays the contents of all storage locations. (See "Storage Ripple Store and Display Functions" in Section 1 of this chapter.)

The ripple tests may be used as a quick confidence test of CE and storage operation or as a means of identifying, with the parity-check indicators, a failing major functional unit within the CE. The overall ripple test routine is contained in 7201-02 FEMM, Form SFN-0203, and is also discussed under "Storage Ripple Store Function" in Section 1 of this chapter.

DIAGNOSTIC PROGRAMS

For a list and discussion of the diagnostic programs available for the 9020D and E, refer to the 7201-02 FEMM, Form SFN-0203.

MARGINAL CHECKING

The marginal checking facility enables several units in the system to be operated with nonstandard voltage conditions, thus providing a means of detecting critical circuits that are deteriorating to a critical voltage-sensitive operating point. In addition, the CE may have its clock period decreased from 200 ns to 195 ns, thus providing a means of detecting circuits that have developed a slower switching speed.

Several of the power supplies in the CE may be varied from the nominal output voltage by controls at the CE. The CE power supplies are varied directly from the system control panel. (The adjustment procedure for these controls is contained in 7201-02 FEMM.)

The marginal checks are performed by running the ROS tests and FLTs with all 6V marginable power supplies in the CE reduced to 5.5V dc and the ROS power supply reduced to 80% of nominal. If the tests passed, rerun them with normal voltages but with the FREQUENCY ALTERATION switch set. After a successful run of the ROS tests and FLTs, run selected diagnostic programs with the marginable power supplies in the CE, storage units, and channels set from nominal to higher or lower output.

SECTION 3. DE FORCE REQUEST AND DE WRAP OPERATIONS

This section discusses the operation of the DE Force Request and DE Wrap functions of the Diagnose instruction and their use in the on-line testing of DEs.

GENERAL DESCRIPTION

- Permits on-line testing of DEs.
- Initiated by CE, using Diagnose instructions.
- Force Request simulates CVG requests.
- Wrap returns CVG data to CE for verification.

DE Force Request and DE Wrap operations allow the diagnostic programmer or maintenance personnel to check DE operation in an on-line environment. Both are initiated at a CE by issuing the Diagnose instruction with the CE in state zero and the proper MCW bits set to 1. The DE must be in state zero, and the CE, DE, and display equipment interfaces must be properly configured for these operations to be successful.

The Force Request operation is used to test the priority logic and display equipment controls in the DE by simulating display equipment requests for data. The Wrap operation permits the verification of display data and display equipment addresses by "wrapping" them back to the CE from the DE. In both operations, the display equipment action is simulated and the display equipment is therefore undisturbed.

Figure 4-9 shows a simplified overall view of the relationship between the CE, DE, and display equipment and of the manner in which Force Request and Wrap operations are implemented. The display equipment consists of up to eight attached display generators (DGs), each of which is associated with up to six character vector generators (CVGs). Only four DGs are shown for simplification since only four may be configured and operating at any one time. Each CVG is associated with a plan view display (PVD) and requests display data from the DE for that display.

In normal operation, up to four DGs, each with up to six CVGs, are configured to four data buffers, A through D. Display data is placed in the DE via the normal display data interface (SDBI). A control area of storage is associated with each data buffer and contains one octword (eight words) for each of six CVGs configured to that data buffer. This control area is initialized by the program and directs each requesting CVG to the correct area of display storage. The octword CVG control areas provide other information as well, including a test byte which the DE automatically sets to all 1's to service a request from the associated CVG. This byte may be tested by the program to determine whether the DE has handled a request from a particular CVG.

Figure 4-9 shows 24 CVG Request lines entering the priority circuitry of the DE through OR circuits (48 lines exist, but only 24 at a time originate from configured CVGs). Also entering the OR circuits are the Force Request latch outputs and the Wrap Request lines. The Force Request latches may be turned on through execution of a Diagnose instruction in the CE; the latches permit testing of the priority circuits in the DE. The diagnostic program may check that these simulated requests have been handled by examining the CVG control areas to determine whether the test byte has been set to all 1's. The Force Request latches may be turned off through execution of another Diagnose instruction at the CE.

Wrap Requests may be generated at the CE via the Diagnose instruction also. Note that these requests do not latch up in the DE. Note, too, that the Wrap Requests also gate the Wrap Bus so that data intended for CVGs as a result of honoring the Wrap Requests is diverted to the CE where it is set into the K-register.

The display equipment is not disturbed during Force Request or Wrap operations because these operations can be executed only with the DE in state zero, which disables the display equipment interfaces.

Details of Force Request and Wrap operations are given in the following text.

DE FORCE REQUEST OPERATION

- Up to 48 CVG requests can be simulated, but only 24 are effective due to configuration limitations.
- Requests are continually presented to DE priority circuits until reset.
- No data is transferred from the DE.

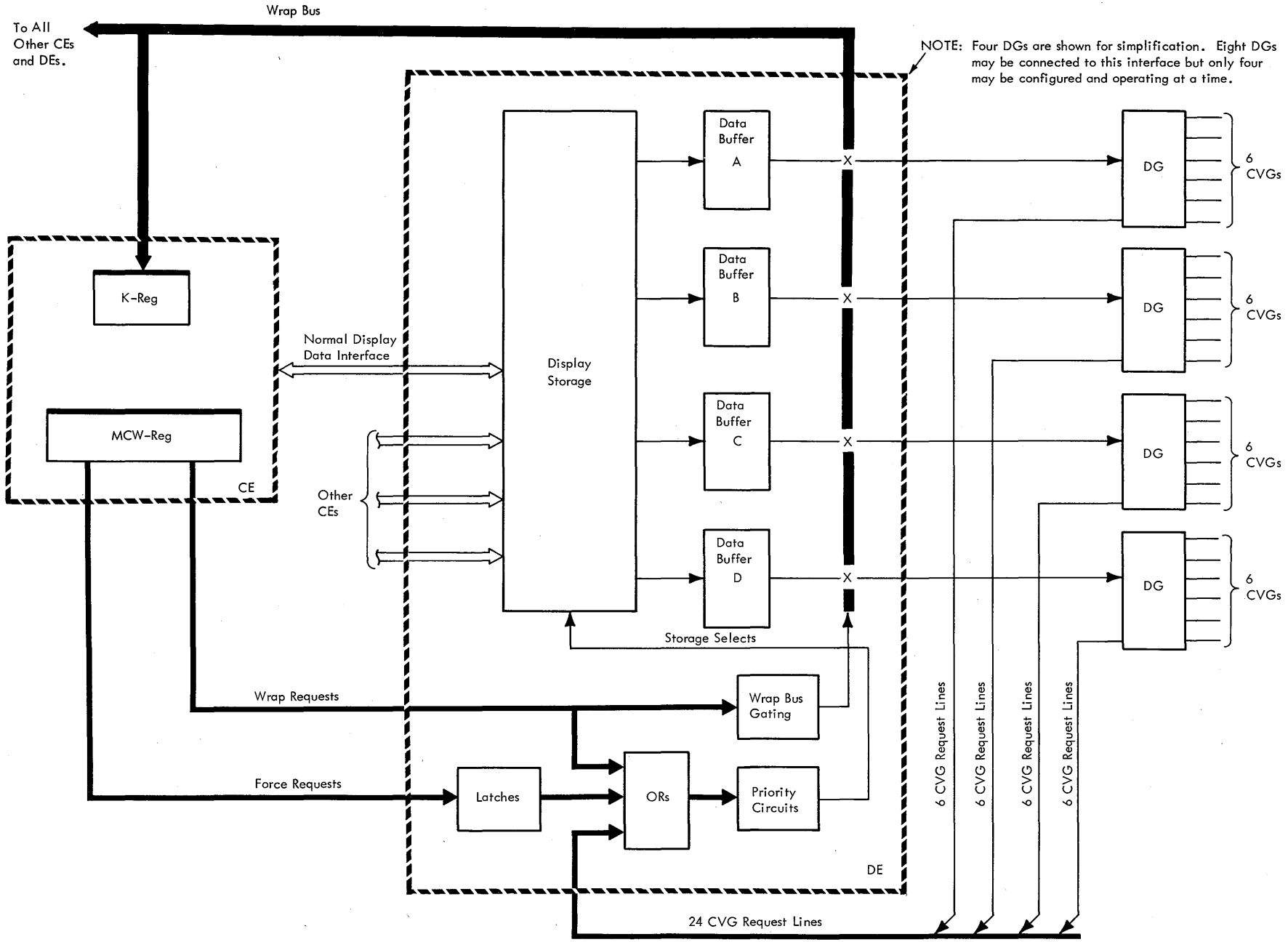


Figure 4-9. DE Force Request and DE Wrap, Simplified

The Force Request operation is initiated at the CE by execution of a Diagnose instruction with MCW bit 47 set to 1. Three fields in the MCW select the CVG or CVGs for which simulated requests are to be generated:

DE Selected, MCW(32–35): This field is coded with one of the hexadecimal DE identifiers from 6 to A to select a particular DE.

DG Selected, MCW(36–39): This field is coded with one of the hexadecimal digits 0 through 8. The digits from 1 to 8 specify DGs 1 to 8; 0 specifies all eight DGs.

CVG Selected, MCW(40–45): This six-bit field corresponds to the six CVGs attached to the selected DG with the leftmost bit corresponding to CVG 1. Any combination of the six CVGs may be selected by setting the bits for the desired CVGs to 1.

It is possible to generate 48 requests by setting the DG Select field to all 0's (select all eight DGs) and the six CVG Select bits to all 1's. Only 24 of these requests can be effective, however, because only four of the eight DGs can be configured at any one time.

As stated previously, the forced requests are latched in the DE and are continually serviced in priority order until reset. Resetting is accomplished by issuing another Diagnose instruction in which the same DE is selected, all DGs are selected (DG Selected field = 0), and no CVGs are selected (all CVG Select bits set to 0). Note that the Force Request latches in the DE are not reset by System Reset.

The forced requests are handled by the DE as if they were normal CVG requests except that the display data is ignored once it is read out; i.e., the data is neither transferred across the DE-DG interface nor to the wrap bus.

Two applications of Force Request are:

1. A DE logout can be forced by (1) placing bad parity data in the DE display storage via Diagnose and (2) specifying a Force Request operation with another Diagnose.
2. A check can be made to determine whether all CVG requests are handled by the DE. First, all 0's are stored in the test bytes of each CVGs control octword. A Force Request is then specified, using the Diagnose instruction. The program then delays, allowing the DE sufficient time to service all the requests. If DE operation is correct, the test bytes for all CVGs serviced will contain all 1's.

DE WRAP OPERATION

- Any combination of six CVGs attached to any one DG may be selected for Wrap operation.
- Data transferred to K-register in CE via Wrap bus and subsequently placed in storage by microprogram.
- Count field in MCW specifies number of quadwords transferred.
- Hardware in CE controls CE clock, checks CVG address, and controls Wrap data gating per timing signals received from DE.

The Wrap operation allows the verification of (1) the data normally sent to a DG by a DE and (2) the address of the selected CVG. The display data takes the normal path through the DE circuitry but is diverted to the DE Wrap bus instead of being gated to the DG. From this bus, the CE stores the data in an SE via K- and ST-registers. The data can then be refetched from the SE and compared with data intended for the selected CVG. If the data received is identical with the data sent, the data, data path, and addressing circuitry for the selected CVG(s) in the DE are functioning correctly.

A DE Wrap operation is initiated at the CE via the Diagnose instruction. To perform a Wrap operation, bit 51 of the MCW must be set to 1. The same three MCW fields are used to select CVGs as are used in the Force Request operation. However, only one DG may be selected at a time. Thus, wrap requests can be generated for up to six CVGs, using one Diagnose instruction.

A count field, MCW(21–31), is used to specify the total number of quadwords to be transferred to the wrap bus as a result of all of the requests called for in the CVG Selected field. Because the count field actually counts halfwords, the low-order bits (29–31) must be made 0 to specify quadwords. For example, if two CVGs were selected and two quadwords were to be transferred as a result of each request, a quadword count of four would have to be specified in the count field. Thus:

MCW Bit:	21	22	23	24	25	26	27	28	29	30	31
Count:	0	0	0	0	0	1	0	0	0	0	0

This is a count of 32 halfwords or, ignoring bits 29–31, 4 quadwords.

The Wrap operation terminates when the counter has been decremented to zero. If no data is received from the DE as a result of a malfunction, a 50-ms timeout occurs (KU 631-KU635) and the counter is decremented. If data is never received, the counter is decremented every 50 ms until it goes to zero, at which time the Diagnose is terminated. If the Diagnose is terminated as a result of wrap timeout, condition code 1 is set to alert the program.

To understand the CE's action during a wrap operation, one must be familiar with the format and timing of the data appearing on the DE wrap bus. The transfer of one quadword (four words) of data and the CVG address to the CE via the DE wrap bus is accomplished in 3.6usec. This time is divided into four 900-ns cycles: a setup cycle, an address cycle, and two data cycles. The sequence chart in Diagram 6-201, FEMDM, shows these cycles and the data on the bus at the different times.

Two lines, 'sample' and 'sync', are timing pulses from the DE. Another line, 'address', is used to serially send the three CVG address bits plus parity from the DE to the CE via the DE wrap bus. The DE follows a set sequence when addressing the CVGs. Thus, there may be cycles when no data will appear on the wrap bus, depending on which CVGs are selected by the wrap operation. Diagram 6-201 shows the circuitry that gates the CVG address line (serial address) into the CE, compares it with the address specified in MCW(40–45), and, if a good address is received, gates a quadword of data (one halfword at a time) from the DE wrap bus into the K-register.

The 'wrap inhibit osc.' latch stops the CE clock until the 'data' latch is turned on. This synchronizes the wrap microprogram (Figure 4-10) with the data received from the DE; i.e., the microprogram does not begin until the 'data' latch is turned on. After this latch is on, the CE clock is stopped at the beginning of each DE sample pulse to compensate for the difference in timing between the clock and these pulses. One CE clock cycle is 200 ns; one sample pulse cycle is 225 ns.

The CE uses the pulses received from the DE over the 'sample' and 'sync' lines to develop CE sample pulses 'zero' through 'three' which gate a CVG address from the 'address' line into three address latches and a parity latch. The address (1 through 6), binarily decoded from these latches, is compared with MCW bits 40–45 (bit 40 is CVG address 1; bit 41, CVG address 2, etc). The three address latches also feed a parity-predict circuit which determines whether the address parity bit should be a 1 or 0. The result of the prediction is used to check the state of the 'address parity' latch. If the compare circuits indicate that a good address and good parity have been received (the MCW bit corresponding to the decoded address is a 1 and the 'address parity' latch equals the predicted parity), the 'data' latch is turned on to begin gating data from the wrap bus into the K-register. Figure 4-10 shows the operation of the wrap microprogram. The microprogram accepts the data from K and stores it, a doubleword at a time, until the specified number of quadwords have been stored.

The CE 'sample' pulses gate the data into K. The 'data' latch, ANDed with DE 'sample' pulses, keeps the CE clock synchronized with the data on the bus by turning on the 'wrap inhibit osc.' latch. This allows the microprogram to gate the wrap data one word at a time (as K is filled), into ST. The data is then stored in an SE (a doubleword at a time) at the storage location specified by D. This location is immediately after the MCW of the Diagnose instruction which initiated the wrap operation. The address sequencer, set to the value of the quadword count in the MCW during the Diagnose instruction, is checked for a value of zero after a complete quadword is stored. When the sequencer reaches zero, the microprogram terminates the instruction and the wrap operation is ended.

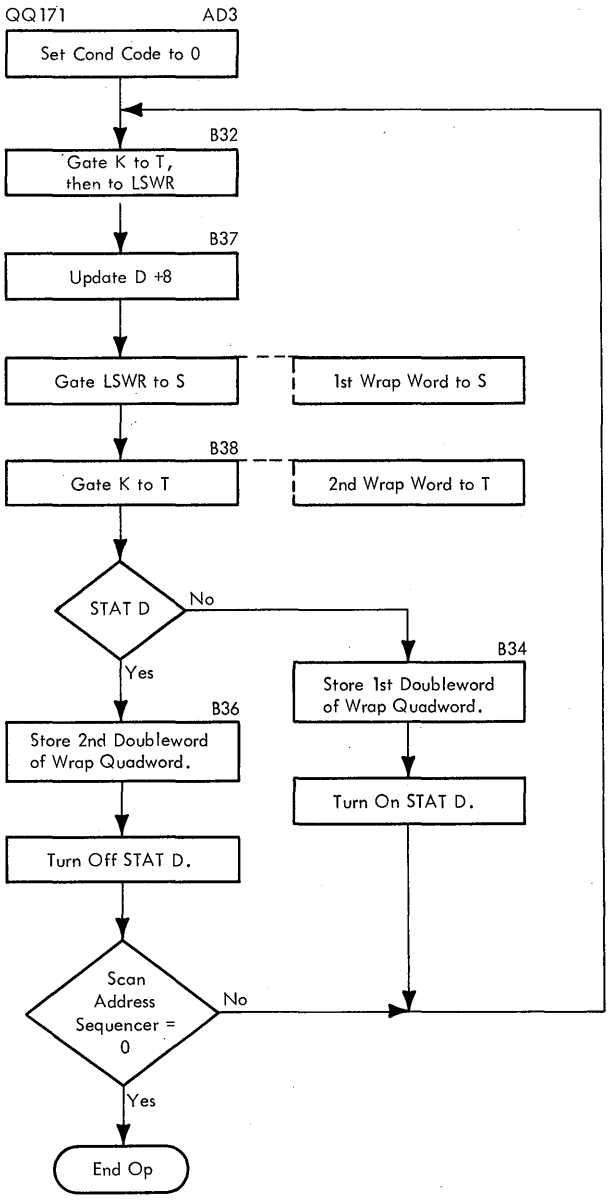


Figure 4-10. Wrap Operation Microprogram

APPENDIX A. SPECIAL CIRCUITS

The only special circuits in the CE are in ROS, for the differential (sense) amplifiers, sense latches, and array drivers. These circuits are described in Chapter 2, Section 2 of this manual. For a discussion of the standard SLT circuits, refer to SLT Component Circuits, FETOM.

The CE executes the EXC control program and is therefore the controlling element in the overall system. Because of this overall system control, the CE is in direct communication with each of the major system elements. The communication takes many forms, including data, controls, and indications.

All of the signal lines or buses that enter or leave the CE are listed on the following pages. Along with each line or bus is a short explanation of its basic purpose or function. Additional information on interfacing may be found in the System Introduction manuals.

CE-CE INTERFACING

The CE is the only element that directly communicates with other elements of its kind. One set of distributed simplex lines originates in a given CE, and three identical sets are received from the other CEs. Also, three sets of simplex lines originate in a given CE, and three identical sets are received.

The interface intercommunication is concerned primarily with direct control operations, which include data communication (one byte) and external CE starts and logouts. In addition to the direct control function, there are controls for reconfiguration and element check monitoring.

Control Bus: This 36-bit bus is used for both configuration control and ATR assignment.

Configuration control (established by the configuration mask) is used to define which elements constitute a subsystem and to avoid interference between subsystems. The configuration control registers in the various elements contain positions to define the other elements with which any given element may communicate at a given time. The CCR positions, or bits, enable or inhibit data and control paths between elements. Refer to the chapter on Configuration Control.

ATR assignment establishes a correlation between logical addresses referred to by the program and actual storage elements within the system.

System Reset (1, 2): A double-railed signal (two lines) which performs hardware resets. The reset is not gated by the CCR. All bits in the CCRs of major elements not in test are reset to 0's except the SCON bits, which are set to 1's.

Element Check (ELC): The 'element check' signal is sent to all other CEs within the system. This signal may result from a CCR or ATR parity error, certain PSBAR stepping situations, and certain hard stops or CE error conditions.

Direct Out Lines: During execution of a write direct CE-to-CE data communication, this bus represents eight data bits and one parity bit fetched from a storage element. This byte of data remains as static signals until the next Write Direct instruction is executed.

Signal Out Lines: This set of five lines is used in conjunction with the 'direct out' lines. The five commands that follow are sent on these lines:

- CE External Start command: This command is issued by a Write Direct instruction and causes the receiving CE to start execution after it obtains a new PSW from location 00000 of its PSA. The receiving CE must be properly SCONed to the sending CE to perform the external start operation.
- CE External Stop command: This command is issued by a Write Direct instruction and causes the receiving CE to perform an element reset and go to the stopped state with the MANUAL light on. The receiving CE must be properly SCONed to the sending CE to execute the external stop operation.
- CE Logout command: This command is issued by a Write Direct instruction to cause the receiving CE to initiate logout procedures. To perform the logout, the receiving CE must be properly SCONed to the sending CE and must now have machine checks masked off in the current PSW.
- CE Write Direct command: This command is issued by a Write Direct instruction and indicates a data-communication-type operation between the sending and receiving CEs. The command causes an external interrupt at the receiving CE by setting a unique bit in its external interrupt register, provided the receiving CE is properly configured (CCR 20–23) to listen to the originating CE.
- CE Read Direct command: This command is issued by the Read Direct instruction and indicates that a byte of data has been taken from the direct data bus. The command causes an external interrupt at the addressed CE by setting a unique bit in its external interrupt register. The receiving CE must be properly configured (CCR 20–23) to the originating CE.

Reconfigure Select: This line, carrying a 5.0-usec pulse from the CE to another CE, causes the CE to set into its configuration register the mask on the output bus. The CE will honor the select if the selector's SCON bit is on in the receiving CE's CCR.

SATR Select: This select line signals and conditions the selected CE to receive the address translation assignment

mask on the 36-bit control bus. Three select pulses are required, one for initial selection and two for transmission of ATR 1 and ATR 2.

SCON/SATR Response: This line is sent to the CEs in response to configuration select, provided the select was honored and the CCR parity is correct. This response signal is also used as a SATR response to acknowledge that the receiving CE was properly SCONed and that the ATR 1 and 2 assignment masks were properly received.

CE – SE INTERFACING

All main storage is located in storage elements (self-contained units, electrically remote from the computing element). Because these SEs are self-contained units, a complete set of buses and interlocking control signals must be provided to ensure proper synchronization and data transfer.

CE to SE Interface

The interface from the CE to an SE involves the following distributed simplex buses: storage data bus in (SDBI), storage address bus (SAB), mark bus, inkey bus, three logword number lines, and the two-line system reset. In addition, two groups of control lines are involved: one group is distributed simplex; the other is simplex. A description of the individual buses and lines follows.

Storage Data Bus In: This group of 72 lines carries data and configuration information from one CE to up to five SEs. The accessed SE gates its receivers to accept the signals from the bus. During a SCON operation, the CE transmits the SCON mask over the SDBI.

Storage Address Bus: This bus consists of 19 lines labeled 1–19 and a parity bit for bits 1–5. Bits 1–4 specify the particular SE and are called the 'box tag'. Thus, the P(1–5) bit is sometimes referred to as P_T, for tag parity, even though bit 5 actually specifies high or low storage. Parity for the address lines (bits 6–19) is sent separately on simplex lines labeled P_A and P_B.

Mark Bus: This bus consists of eight bits (0–7) plus a parity bit and represents the store or regeneration control lines for each of the eight bytes of the doubleword accessed by the CE.

In Key Bus: This bus consists of five bits (0–4) plus a parity bit. The high-order four bits are compared with the key bits stored in the storage protection area of the SE. The low-order bit (bit 4) indicates that fetch protection is active.

Logword Number: Three lines carry the logword number from the CE to an SE. The lines gate the logwords onto the storage data bus out.

System Reset (1, 2): These signals cause a storage reset when a system reset occurs. In addition to resetting the SE itself, the storage reset causes the SE's CCR to be reset to all 0's (except for parity and SCON bits, which are set to 1's). These reset lines are not gated by the CCR.

Set Key: This signal from the CE causes an SE to perform a set-key cycle, i.e., to set the protection key from the CE into the storage protect array.

Insert Key: This signal causes the SE to place the key from the storage protect array on the out key bus.

Store: This is a control line which permits the SE to perform a data store.

Test and Set: This signal causes the SE to perform a test-and-set storage cycle in which a doubleword is fetched and regenerated into storage (except for the addressed byte, which is set to all-1's before being returned to storage).

Cancel: This signal causes the SE to regenerate the data fetched from storage without transferring it to the user element.

Defeat Interleave: This signal indicates to the SE that the DEFEAT INTERLEAVE switch on the CE is active or the CE has been placed in Defeat Interleave mode via a Diagnose instruction. While the CE reverses SAB bits 6 and 20 to defeat interleaving, the IOCE does not. Therefore, the 'defeat interleave' signal is sent to enable the SE to reverse the roles of IOCE address bits 6 and 20. This is accomplished by (1) substituting bit 20 for bit 6 when gating the IOCE storage address buffer to SESAR and (2) sampling bit 6 (bus bit 14) to determine whether the access request is for the odd or even half of storage.

Defeat Interleave and Storage Reverse: This signal has the same significance as 'defeat interleave' except that IOCE address bit 6 (bus bit 14) is inverted before it is used for selecting odd or even storage.

Address Compare Sync: This signal provides a negative significant sync pulse for scoping when the address switches on the CE control panel match the SAB and the ADDRESS COMPARE switch is in the normal position.

CE Power On: This signal is raised during CE power on reset and falls to ground level before power goes off. It inhibits the output of the associated communication bits in the CCR of the SE.

Double Cycle: This signal guarantees two sequential even or odd storage cycles by inhibiting priority scanning in the SSU until the second select from the issuing CE. The CE must issue the second select within the SE timeout period or the SE will issue a pulse ELC, reset the priority circuitry, and become available for new requests. The SE does not wait for 'logout stop' and does not reset outstanding selects.

Normal Op: This signal ensures that control bus driver or receiver failures do not cause multiple operation execution, resulting in lost data. The 'normal op' signal is valid with fetch and store operations only. If it is not sensed with

fetch or store; or if it is sensed with 'test and set', 'set key', 'double cycle', 'suppress log check', or 'insert key'; an address check is issued to the using CE, the entire SE stops, ELC is issued to all CEs, and the SE waits for 'logout stop'.

Select Even: This signal is sent to an SE to request an even storage cycle.

Select Odd: This signal is sent to an SE to request an odd storage cycle.

Logout Stop: This signal sets the 'SE stopped' latch in the SE, causing the SE to halt all activity at the end of the cycle in progress (if not already stopped). The SE issues 'SE stopped' to the using elements and a level ELC to all CEs. It then remains stopped until logged out, reset, or reconfigured.

Logout Select: This signal is sent to the SE to request a doubleword of logout data. The signal is used in conjunction with the three 'logword number' lines, which specify the doubleword to be transferred.

Logout Complete (Check Reset): The CE sends 'logout complete' to the SE at the completion of a logout and during a subsystem reset. The SE senses this signal as a check reset, and storage control logic is reset. At the completion of the reset sequence, the 'SE stopped' latch is reset.

Reconfigure Select: This signal causes the SE to gate portions of the SDBI into its CCR, provided the SE is properly SCONed to the issuing CE. If a CCR parity error exists in the SE when 'reconfigure select' is received or when no SCON bits are on and the SE is not in state 0, CCR gating is ignored, and the SE accepts the SCON data.

Suppress Log Check: This signal suppresses 'data check' and its associated ELC signal.

P_A and P_B: Parity conversion circuitry in the CE develops the P_A and P_B parity bits for 14 of the SAB bits (bits 6–19) in two groups of seven bits each: P_A for bits 6–12 and P_B for bits 13–19. This parity generation takes into account the bit-20 and bit-6 reversal involved in 'defeat interleave' and the bit-6 inversion for 'storage reverse'.

Because generation of the parity introduces several nanoseconds of delay, the P_A and P_B are sent separately to each SE via simplex lines rather than with the rest of SAB (which uses a distributed simplex bus).

SE to CE Interface

The SE to CE interface comprises the storage data bus out (SDBO), the out key bus, and two groups of control lines. The buses and one group of control lines are multiple-driver simplex lines. The remaining control lines are simplex. A description of the SE to CE interface buses and lines follows.

Storage Data Bus Out: This is a 72-bit bus that carries eight 8-bit bytes and the associated eight parity bits. The eight bytes consist of data from a normal fetch operation or logout information during a storage logout operation.

Out Key Bus: This bus consists of five key bits and a parity bit. The high-order four bits represent the storage key; the low-order bit is the fetch-protect bit. The bus carries the key to the CE during an insert key operation.

Advance SDBO: This is a single multiple-driver simplex line sent in advance of data on a fetch operation and together with data on all other storage cycles. It is used by the CE as a signal to sample for errors.

Advance Keys: This is a single multiple-driver simplex line activated by the storage-protect feature in the SE during an 'insert key' cycle to allow the CE to ingate the key from the 'out key bus'.

Protect Check: This is a single multiple-driver simplex line activated by the storage-protect feature of the SE when the protection key and the storage key do not agree during a normal store or fetch operation.

The following control lines are simplex lines:

Accept: This signal indicates that the SE has received a 'select' from the CE and has started the storage cycle.

Logout Advance: 'Logout advance' is sent to alert the CE that logout data is available on the SDBO.

Element Check (ELC): ELC may be a pulse or a level. It is sent to all CEs, regardless of configuration, when an error or other abnormal condition occurs at the SE.

Pulse ELC is coincident with CCR parity error, temperature out of tolerance check (OTC), 'on battery' signal OBS, and 'storage check', 'address check', or 'data check'.

Level ELC is coincident with overvoltage or overcurrent condition, power off or power check, and 'SE stopped'.

Reconfigure Accept: This signal indicates to the CE that the SE has loaded the reconfiguration data from the SDBI into the CCR and that good parity exists.

SE Stopped: This signal is issued to all configured CEs to indicate that the SE has stopped in response to a 'logout stop' (LOS) signal from a CE or IOCE. This signal inhibits all operations except logout, reconfiguration, and reset.

Logout SE Stopped: This simplex line has the same timing as 'SE stopped' except that it is not degated in Diagnose Logout mode. It enables the CE to store the final word of the SE logout and allows correct operation of CE SCI logout controls after the rise of 'logout complete'.

SE Ready: This signal indicates to the using element that the SE is available; i.e., the SE has power up, is not in test, is properly configured, and is not being reset.

SBO Gate: This signal is used by the CE to identify the SE.

Address Check: This signal is activated by the SE when any of the following addressing errors occur: mark parity, address parity, tag parity, tag mismatch, multi-accept condition, storage-protect parity, in-key or out-key parity or an invalid op error. This signal is always accompanied by a pulse ELC.

Data Check: This signal is activated by the SE when it detects a storage data parity error unless inhibited by 'suppress log check'.

CE-DE INTERFACING

The CE-DE interface has three major purposes:

1. Allows data transfers for store and fetch operations and for reconfiguration.
2. Simulates the request capability of 24 CVGs so that the priority circuitry in the DE can be tested.
3. Allows data normally transferred to the CVGs to be returned to the CE via the wrap bus for validity checking.

Most of the interface lines are the same as the storage element interface and are used in the same way to allow the DE to perform as a storage device. Additional interface lines implement the priority and CVG data-checking facility. The additional lines consist of 12 lines from the CE to the DE and a 19-line bus from the DE to the CE, as follows:

CE to DE:	
CVG REQUEST	(6 lines)
DG Selected	(4 lines)
Wrap	(1 line)
Set Force Request	(1 line)
DE to CE:	
Wrap Bus	(19 lines)

CE to DE Interface

The CE to DE interface involves a number of distributed simplex buses and control lines as well as a group of simplex buses and control lines. All distributed simplex lines are positive at the active level except for 'system reset'. The simplex lines are negative at the active level except for ELC. A description of each bus and control line follows.

Storage Data Bus In: A group of 72 lines carrying data and configuration information from one CE to one of four DEs. The accessed DE gates its receivers to accept the signals from the bus. During a SCON instruction, the DE gates 27 bits of data plus 4 parity bits from the SDBI into its CCR.

Storage Address Bus (SAB): The SAB consists of 19 lines labeled 1–19 and a parity bit for bits 1–5. Bits 1–4 specify the particular DE and are called the "box tag". For this reason, the P(1–5) bit is sometimes referred to as P_T, for tag parity, even though bit 5 is not included in the tag. Parity for the address lines (bits 6–19) is sent separately on simplex lines labeled P_A and P_B.

Mark Bus: This bus consists of eight bits (0–7), plus a parity bit, and represents the store or regeneration control lines for each of the eight bytes of the doubleword accessed by the CE.

In Key Bus: This bus consists of five bits (0–4) plus a parity bit. The five bits are compared with the key bits stored in the storage-protection area of the DE.

Logword Number: Three lines carry the logword number from the CE to a DE. The lines gate the logwords onto the 'storage data bus out' under CE control. They are always at the zero state when not in use.

System Reset (1, 2): These signals cause a storage reset when a system reset occurs. In addition to resetting the DE itself, the storage reset causes the DE's CCR to be reset to all-0's (except for parity and SCON bits, which are set to 1's). These reset lines are not gated by the CCR.

CVG Request (Six lines): These lines are used by the CE to simulate CVG requests. They are made active by the CE executing a Diagnose instruction and remain active until reset by the CE.

DG Selected (Four lines): These lines are used by the CE, executing a Diagnose instruction, to select any combination of four of the eight DG interfaces in the DE. The 'DG selected' lines are used in conjunction with 'wrap' and 'set force request', all under control of the Diagnose instruction.

Set Key: This is a signal from the CE, causing a DE to perform a 'set key' cycle; i.e., to set the protection key from the CE into the storage protect array.

Insert Key: This signal causes the DE to place the key from the storage protect array on the 'outkey bus'.

Store: This is a control line that permits the DE to perform a data store.

Test and Set: This signal causes the DE to perform a test and set storage cycle in which a doubleword is fetched and regenerated into storage (except for the byte specified by the R1 field, which is set to all-1's before being returned to storage).

Address Compare Sync: This signal provides a negative significant sync pulse for scoping when the address switches on the CE control panel match the SAB and the ADR COMPARE switch is in the normal position.

CE Power On: This signal is raised during CE power-on reset and falls to ground level before power goes off. It inhibits the output of the associated communication bits in the CCR of the DE.

Double Cycle: This line is not used at the DE except to check 'normal op'.

Normal Op: This signal ensures that control bus driver or receiver failures do not cause multiple operation execution, resulting in lost data. The 'normal op' signal is valid with fetch and store operations only. If it is not sensed with fetch or store; or if it is sensed with 'test and set', 'set key',

or 'insert key'; an 'address check' is issued to the using CE, the entire DE stops, ELC is issued to all CEs, and the DE waits for 'logout stop'.

Wrap: By executing a Diagnose instruction, the CE uses the 'wrap' signal in conjunction with the 'DG selected' signals to gate one of the four DG/DE interfaces to the 'wrap bus' and to initiate 'wrap' requests.

Set Force Request: This signal is used by the CE, executing a Diagnose instruction, to initiate simulated CVG requests.

Select Even: This signal is sent to a DE to request an even storage cycle.

Select Odd: This signal is sent to a DE to request an odd storage cycle.

Logout Stop (LOS): This signal sets the 'DE stopped' latch in the DE, causing the DE to halt all activity at the end of the cycle in progress (if not already stopped). The LOS signal must be issued by a CE in conjunction with the first 'logout select' to establish logout priority; it must remain active throughout the logout procedure to retain priority. Upon receipt of LOS, the DE issues 'DE stopped' to the using elements and a level ELC to all CEs. It then remains stopped until logged out, reset, or reconfigured.

Logout Select: This signal is sent to the DE to request a doubleword of logout data. The signal is used in conjunction with the three 'logword number' lines, which specify the doubleword to be transferred.

Logout Complete (Check Reset): The CE sends 'logout complete' to the DE at the completion of a logout and during a subsystem reset. The DE senses this signal as a check reset, and storage is reset. At the completion of the reset sequence, the 'DE stopped' latch is reset.

Reconfigure Select: This signal causes the DE to gate portions of the SDBI into its CCR, provided the DE is properly SCONed to the issuing CE. If a CCR parity error exists in the DE when the 'reconfigure select' is received or no SCON bits are on and the DE is not in state 0, CCR gating is ignored and the DE accepts the SCON data.

P_A and P_B: Parity conversion circuitry in the CE develops the P_A and P_B parity bits for 14 of the SAB bits (bits 6–19) in two groups of seven bits each. P_A is the parity bit for bits 6–12; P_B is the parity bit for bits 13–19.

Because the generation of the parity introduces several nanoseconds of delay, P_A and P_B are sent separately to each DE via simplex lines rather than with the rest of SAB (which uses a distributed simplex bus).

DE to CE Interface

The DE to CE interface consists of a number of buses and control lines which are multiple-driver simplex; i.e., drivers in up to four DEs share the bus to a given CE. In addition, a group of control signals are sent via simplex lines. Individual buses and lines are described below.

Storage Data Bus Out (SDBO): The SDBO consists of 72 lines (64 data bits and 8 parity bits) which are used to transfer data from the DE to the CE during a fetch or logout operation.

Out Key Bus: This bus consists of five data bits and a parity bit. The bus is used to transfer the key bits from the storage protect array in the DE to the CE during an insert key operation. It is multiple-driver simplex.

Wrap Bus: This bus is a 19-bit-wide, multiple-driver/multiple-receiver simplex bus used to return or wrap data, which normally goes to CVGs, to the CE for checking purposes. This path is also used to check the DE/DG interfaces and DE control logic without the use of the DG. The first two bits are for timing; these are the 'data sample timing' line and the 'word sync timing' line. The third bit is the 'CVG address' line. The remaining 16 lines (labeled 0–15) constitute the data bus itself.

Advance SDBO: Advance SDBO is a single multiple-driver simplex line that is activated by the DE in advance of data out on the SDBO for all storage cycles.

Protect Address Check: This is a multiple-driver simplex line that is activated by the storage-protect portion of the DE to indicate to the CE that the keys do not match during an attempted store or an attempted fetch from a fetch-protected location.

The following control lines are simplex:

SBO Gate: This signal is used by the CE to identify the source of data on the SDBO during a fetch cycle.

Accept: This signal indicates that storage has been started in response to the CE's request (select).

Logout Advance: This signal is sent to the CE to indicate that logout data is valid on the SDBO.

Element Check (ELC): This signal is sent to all CEs (regardless of configuration) to alert them that an abnormal condition has occurred. Unlike other simplex lines from the DE, the ELC signal, which may be a pulse or a level, is positive when active.

Pulsed ELCs are coincident with:

1. CCR parity error
2. Temperature out of tolerance (OTC) condition
3. On-battery supply (OBS)
4. Logic check:
   - Data parity check
   - Address, mark, key, or box-tag parity check
   - Box-tag mismatch
   - Normal-op check
   - Multi-accept check
   - Refresh parity check
   - DG data register parity check
   - Local store parity check

Level, or static, ELCs are coincident with:
1. Power check or power down
2. DE stopped

Reconfigure Accept: This signal indicates to the CE that the DE has loaded the CCR with reconfiguration data from the SDBI and that correct parity and no redundancy check exists. A redundancy check results when an attempt is made to configure more than one data register to a DG register, or vice versa. A redundancy check inhibits 'reconfigure accept' but does not cause an ELC.

DE Stopped: This signal indicates to all configured CEs that the DE is stopped for logout. It results from the setting of the 'DE stopped' latch, which inhibits all operations except logout and reconfiguration.

DE Ready: This signal indicates to the CE that the DE is available for use. Specifically, it indicates the DE has power on, is not in test, is not configured away from the particular CE, and is not being reset.

Storage Data Check: This simplex line indicates a storage data parity error to the CE. It is accompanied by a pulse ELC to all CEs.

Storage Address Check: This is a simplex line which indicates to the CE that one of the following errors has occurred: mark parity, address parity, key parity, box-tag parity, box-tag mismatch, or multi-accept. All of these are associated with storage addressing. A storage address check is accompanied by a pulse ELC to all CEs.

CE-IOCE INTERFACING

The 9020D and 9020E systems include a maximum of three IOCEs, each of which controls one multiplexer channel and up to three selector channels. All system communication (except configuration) with I/O units is accomplished through the IOCEs by means of the following I/O instructions:

Start I/O
Test I/O
Halt I/O
Test Channel
Set PCI

Because all I/O operations are initiated by I/O instructions, the CE must have control over the IOCEs. Any CE can control any IOCE. One CE may have several IOCEs on-line (configured) at one time. However, communication between a CE and the IOCEs is in an interleaved manner, allowing an operation between the CE and only one IOCE at a given instant. On the other hand, any IOCE may be under control of only one CE at a time.

During I/O operations, certain control signals pass directly between the CE and IOCE via the CE-IOCE interface; control information is passed between the CE and IOCE via main storage. During execution of an I/O instruction, the CE signals the selected IOCE which channel and unit are to be selected and which of the I/O instructions is to be executed. The CE also transmits a quantity called the preferential storage base address (PSBA) which indicates to the IOCE which preferential area contains the CAW, CSW, PSWs, and other critical control information. the CAW and CSW locations are used for CE-IOCE communication. Upon completion of the I/O instruction, the IOCE transmits a value to be placed in the CE's condition code register. Subsequent CE instructions may be used to test the condition code to determine the result of the I/O instruction.

I/O interruptions are also controlled over the CE-IOCE interface. The CE transmits the system mask portion of its current PSW to the IOCE. The IOCE uses the system mask to gate the channel interruption conditions when signaling its controlling CE of an outstanding interruption. At the end of its current instruction, the CE supplies the PSBA and signals the IOCE to proceed with the interruption. At this time, the IOCE performs the highest-priority pending interruption that is not masked off. After the CSW has been stored, the IOCE stores the associated channel and unit address and a portion of the system mask in the proper location in main storage to partially form the old PSW for the I/O interruption. Upon completion of the channel portion of the interruption, the IOCE signals the CE to complete the interruption.

The CE-IOCE interface is also used to perform other specific functions such as IPL, FLT load, logout, start I/O processor, and direct control.

CE to IOCE Interface

Interfacing for the CE to IOCE includes a control bus and associated signal lines. The control bus transfers information such as configuration masks, ATR assignment masks, PSBAR indications, IPL and FLT operations, I/O instructions (including Start I/O Processor), and direct control commands such as Write Direct Processor Start, Stop, or Interrupt. The control bus is never used to transfer data. Control signals define the information on the control bus and synchronize the operations. The various lines and buses are described below.

Control Bus: This bus contains 36 multiplexed lines: 32 bit positions plus 4 parity bits. It services the configuration control register, address translation register, I/O processor, I/O instructions, IPL, logout, interrupt signals, and FLTs.
1. 'Reconfigure select' and 'SATR select' use the bus to set the configuration control register or the address translation register.
2. I/O instructions use the bus to transmit the channel and unit address, preferential storage base address (PSBA), and the I/O instruction identification.

3. IPL and FLT load use the bus for the channel and unit address and PSBA.
4. Logout uses the bus for PSBA only.
5. 'Permit interruption' uses the bus for PSBA only.
6. 'Start I/O processor' uses the bus to transmit a storage key and an address.

System Reset (1, 2): This is a 1-ms signal which precedes a system IPL. The 'system reset' is sent on two lines. Line 1 must go negative and line 2 must go positive simultaneously to cause the reset. These lines are not gated by the CCR.

When active, the 'system reset' performs hardware and microprogram resets in the IOCE, provided the TEST switch is not on. 'System reset' causes the CCR to be reset to 0's, (except for the SCON bits, which are set to all 1's).

Subsystem Reset (1 line): This line is activated by the CE to cause the IOCE to do an element reset. To be effective, the IOCE must be properly configured to the sending CE. The IOCE does not reset its CCR as a result of 'subsystem reset'.

System Mask Bits 16–19: These lines are sent to all IOCEs and are used to form the first portion of PSW byte 2 on channel interrupts. (See also system mask bits, under simplex lines.)

360 Mode Operation: This signal causes the IOCE to operate in the 360 mode. IOCE 1 must be configured to CE 1.

The following lines are all simplex:

I/O instruction: This line is sent to a selected IOCE. The preferential storage base address, unit address, channel address, and the decoded I/O instruction are placed on the control bus. The I/O instruction line is then brought up to tell the IOCE to take the information from the control bus. The line remains static until a response is received from the IOCE.

FLT Load: This line is not used by the CE in the 9020E system. The 'initial program load' line is used during the performance of an FLT from a CE.

Initial Program Load: This line, generated in the same manner as an I/O instruction, indicates to the selected IOCE that it is to perform an IPL. The control bus contains the preferential storage base address and the unit and channel address when this line is brought up.

Permit I/O interrupt: This line is sent to the IOCE in response to an IOCE I/O interrupt request. This signal allows the IOCE to store its CSW and interrupt code field of the PSW.

Logout: This line is issued from a Write Direct instruction and causes the IOCE to begin a logout operation. The IOCE must be SCONed to the sending CE.

Reconfigure Select: 'Reconfigure select' consists of three lines, one to each IOCE. Each line is generated from the select mask and gates the configuration bits from the control bus into the IOCE. The IOCE must have the issuing CE's SCON bit on.

System Mask: The system mask lines are static lines from the CE's PSW register bits 0–6 and 16–19. These lines mask the channels in the IOCEs and are distributed as follows:

PSW bits 0–3 to IOCE 1
PSW bits 4–6, 16 to IOCE 2
PSW bits 17–19 to IOCE 3
PSW bits 16–19 to all IOCEs

Bits 16–19 are sent to all IOCEs and are used in forming the first portion of PSW byte 2 on channel interrupts. However, these bits (16–19) are sent via distributed simplex lines.

Permit Machine-Check Interrupt Request: A line sent to configured IOCE in response to a machine-check interrupt request. This signal allows the IOCE to continue with the logout.

FLT Backspace: This signal causes the IOCE to branch from its wait loop and backspace the FLT tape over one record. When the operation is completed, an 'FLT complete' signal is returned to the CE.

SATR Select: This line signals and conditions the selected IOCE to receive the new address translation assignment mask on the 36-bit control bus. Three select pulses are required, one for initial selection and two for transmission of ATR 1 and ATR 2.

Write Direct Start: This line causes an IOCE processor to leave the stopped state and go to either the running or wait state. This signal is sent to the IOCE when the CE executes an SIOP instruction. When SIOP is sent, the control bus has data in the format labeled "Start I/O Processor".

Write Direct Stop: This line causes an IOCE processor to go to the stopped state at the end of the current processor instruction.

Write Direct Interrupt: This signal causes an external interrupt of an IOCE processor.

IOCE to CE Interface

Interfacing from the IOCE to the CE is concerned primarily with control-type signals. The control bus, which transfers information from the CE to the IOCE, is strictly a one-way bus and perform no functions in this section. Programming restrictions allow an IOCE to be configured to only one CE at a time. IOCE to CE interface lines are described below.

Condition Code: Two lines sent to the CEs as the result of an I/O operation, indicating a value to be set into the PSW.

SCON/SATR Response: A response sent to the CE after the SCON instruction has been accepted and the IOCE has set its CCR without detecting a parity error in the CCR.

This response signal is also used as a set address translation response to acknowledge that the receiving CE was properly SCONed and that the ATR 1 and ATR 2 assignment masks were properly received.

Response: A line sent to the CE to indicate that the condition code for an I/O operation is present, IPL is complete, I/O interruption is complete (CSW and the interrupt code field of the old PSW are stored), or machine check interrupt is complete (CLU logout).

Check Response: A signal line to the CE, indicating that a parity check has been detected in the IOCE on data from the CE via the control bus.

Reset ROS Timeout: A signal sent to the CE, indicating that the IOCE will process the I/O instruction but is presently processing data. The signal resets the CE's countdown loop to its maximum value, preventing it from timing out.

PSA Lockout: A line indicating that the IOCE tried to access the PSA but did not receive a reply from the storage element accessed or that the PSA access was issued to a logout-stopped SE. This signal causes a program interruption in the CE.

FLT Complete: A response to the CE indicating that the FLT backspace request has been completed.

Write Direct Interrupt: This line is used by the I/O processor to signal the CE to take an external interrupt.

TIC: This signal is sent to the CE to indicate that a Transfer In Channel (TIC) command has been encountered in the channel. It is used by the CE when running FLTs.

Gap: This line is sent to the CE to indicate that the channel has encountered an inter-record gap, i.e., the end of a tape record. The CE uses this signal when running FLTs.

Element Check: This signal is issued by an IOCE as a pulse whenever a parity error is detected in the ATR or an error condition is detected that requires a CLU or selector channel logout.

The 'element check' signal is issued as a level if, during a CLU logout, the IOCE detects a condition that will not permit logout to be performed. An 'element check' signal causes an external interrupt in the CE.

On Battery Signal (OBS): This signal, when issued as a static condition to the CE, indicates that the IOCE is operating on its battery supply. The 'OBS' is issued as a pulse whenever CCR parity is detected during execution of the SCON operation in the IOCE. This signal causes an external interrupt in the CE.

Out of Tolerance Check (OTC): A signal to the CE, indicating that the IOCE temperature-sensing thermals have detected an out-of-bounds temperature. This signal causes an external interrupt in the CE.

I/O Interrupt Request: A signal from the IOCE, informing the CE of status changes in the channels or I/O devices. When a status change is detected by the IOCE and the channel is masked on, an I/O interrupt request is sent to the CE. The channel that requested the interrupt waits for a "permit I/O interrupt" from the CE before storing a channel status word which indicates the reason for the interrupt request.

Also stored are the channel and unit address, IOCE address, mask bits 16–19, and the interruption code in the old PSW. If the channel control check or the interface control check bit is on in the CSW, a selector channel logout has been performed.

Machine Check Interrupt Request: The detection of a CLU error by an IOCE causes the IOCE to stop and request a PSBA for logout via the 'machine check interrupt request'.

A CE in an I/O instruction or interrupt process with the IOCE issuing the machine check interrupt request, terminates the process and proceeds to the next I-fetch. That I-fetch, or any I-fetch, has an exception branch to a special microprogram routine. This routine issues a 'permit machine check interrupt' and enters a timing loop while the IOCE performs a logout. The IOCE holds up 'reset time out' while logging out. The CE waits for a response line or, in the event of further IOCE error, a timeout. In either case, the CE completes a 'machine check interrupt' by storing and fetching the correct PSWs. No logout occurs in the CE. The old PSW contains the IOCE identity in the interrupt code as follows:

	Bit	
IOCE	30	31
1	0	1
2	1	0
3	1	1

A timeout condition should always be accompanied by an IOCE element check. Simultaneous I/O interrupt requests for MC interrupt from multiple IOCEs are serviced in priority of IOCE 3, 2, and 1.

CE-PAM/TCU/SCU INTERFACING

Interface lines between the CE and PAM, between the CE and TCU, and between the CE and SCU, for the most part, represent reconfiguration, system reset, and element checks. Other than these lines, there is no control or data flow directly to or from the CE.

CE-to-PAM/TCU/SCU Interface

Configuration Mask: Only the required 11 positions of the overall configuration mask are sent to the PAMs and TCUs. The items sent include the two state bits, the four-bit SCON field for reconfiguration, and the three-bit field to

define the controlling IOCEs. Two parity bits are used in the transfer.

System Reset (1, 2): The 'system reset' signal results from a system IPL. It causes a hardware reset to the PAM, TCU, or SCU. The reset is not gated by the CCR. All bits in the CCR are reset to 0's (except the SCON bits, which are set to all-1's).

Configuration Select: This line, carrying a 5.0-usec pulse, causes the PAM, TCU, or SCU to set the configuration mask into its CCR. The PAM, TCU, or SCU will honor the select if the selector's SCON bit is on in the receiving PAM, TCU, or SCU's CCR.

PAM/TCU/SCU-to-CE Interface

Element Check: A level simplex signal is sent from each PAM/TCU/SCU to all CEs when one of the following conditions occur:

1. Parity check in the CCR
2. Power failure

Configuration Response: This line is sent to the CE in response to configuration select if the select was honored and the CCR parity was correct.

CE-SYSTEM CONSOLE (SC) INTERFACING

The greater portion of the interface is concerned with sending indication (status) signals from the CE to the SC and control signals from the SC to the CE. The CE cannot initiate any data transfer directly to the SC via the interface. Subsystem configuration indications, for example, which are under program control, are initiated by the CE and handled as a normal I/O operation via the IOCE.

CE-to-SC Interface

Many of the indications from the CE to the SC represent the current status of the CE. The most important dynamic indications include the state of the CE, logic checks, current instruction address, and manual or wait status. The CE to SC lines and buses are described in the following paragraphs.

Data Indicator Lines: These 36 multiple-driver simplex lines provide for the display of four bytes of data from a main or local store location. They originate in the CEs at the T-register.

Load Indicator: This multiple-driver simplex line originates in one or more CEs and turns on a common indicator. The indicator is lit by the respective CE from the time an IPL operation starts until the operation is complete.

Invalid Selection: This multiple-driver simplex line originates in one or more CEs and turns on a common indicator. The selected CE lights the indicator whenever an invalid or illegal storage address is specified by the operator during a manual operation.

Instruction Counter (IC) Indicators: These 27 simplex lines provide data to the SC for the display of the current instruction address at the CE. One set of these lines is indicated at the SC for each CE.

Manual Indicator: These simplex lines originate in each CE and terminate in a unique indicator at the SC to indicate when the associated CE is in the stopped state.

Wait Indicator: These simplex lines originate in each CE and terminate in a unique indicator at the SC to indicate when the wait bit in the associated CE's PSW is set to 1.

State Indicators: These four simplex lines originate in each CE and terminate in unique indicators at the SC. Only one of these four signals will always be active to reflect the present status of the originating CE.

Logic Check Indicator: These simplex lines originate in each CE and terminate in a unique indicator at the SC to indicate that the CE has detected one of its own logic check conditions.

Power Check: This signal line indicates that the temperature in the CE has drifted to within about 10 percent of the shutdown tolerance. The signal also indicates the loss of voltage or a normal power-off (but not an element master power-off) condition.

Battery: This signal indicates that the CE has switched to battery power.

SC-to-CE Interface

All SC controls to the CEs are gated by the system interlock switch, which requires a key operation to activate the logic. SC operations require that the CE TEST switch be off. In addition, all functions going to the CE (except for 'all stop') are further gated in the appropriate CE by a 'select CE' signal originating from the select CE rotary switch on the SC. The interface lines are described in the following paragraphs.

Address Keys: These 24 signal lines (+3 parity) result from the 24 instruction address keys on the SC and provide addressing of any addressable local store or main storage location.

Data Keys: These 32 signal lines (+4 parity) result from the 32 storage data keys on the SC and provide manual data for storing into any addressable local store or main storage location.

CE Select: These four select lines result from the four-position rotary switch at the SC. Proper CE selection is

under switch card control in the receiving CEs. This signal provides the necessary gating in the CEs for all manual operations (except All Stop) issued from the SC. The SYSTEM INTERLOCK switch must be turned on for this select switch to be enabled.

Stop: This signal places the selected CE in the stopped state without destroying its environmental status. The selected CE proceeds to the end of the instruction being executed at the time the stop is initiated. If the current instruction causes a program interrupt, program status words (PSW) will be changed before stopping. An I/O device will be allowed to complete its operation, although I/O or external interrupts will not be recognized.

Start: This signal starts the selected CE. If start is issued after a manual stop, the CE continues as though no stop occurred.

Store Select Main: This signal results from the STORAGE switch being in the MAIN position and causes main storage addressing at the selected CE during either fetch or store manual operations. This signal line is not active with the STORAGE switch in the LOCAL position, causing local storage addressing at the selected CE.

Display: This signal causes the selected CE to place the contents of a storage location, specified by the address keys and STORAGE SELECT switch, onto the data indicator lines to the SC.

Store: This signal causes the selected CE to store the contents of the SC storage data keys in the storage location specified by the 24 address keys and the STORAGE switch.

Set Instruction Counter (IC): This signal transfers the contents of the SC address keys to the instruction counter of the selected CE.

Interrupt: This signal results from depression of the interrupt key and causes a console interrupt signal, which sets bit 25 in the PSW of the selected CE.

Address Compare Stop: This line conditions the selected CE so that any storage access to the address specified in the SC address keys causes the CE to enter the stopped state at the end of the instruction that made the memory reference.

Address Compare Loop: This line causes the selected CE to loop between the address set in the SC address keys and the address set in the storage data keys. When the selected CE makes a storage access to the address specified in the address keys, an unconditional branch is made to the address specified in the storage data keys. Programming can establish a loop condition between the two sets of keys.

Rate: This signal operates with the selected CE and the START and STOP keys on the SC. With this signal, each depression of the START key results in one complete instruction being executed. Any machine instruction can be executed in this mode.

Load: This signal line initiates an IPL in the selected CE.

Load Unit Address Bits: These eight signal lines (+ parity) result from two of the three rotary-type LOAD UNIT switches on the SC. These two hexadecimal characters provide an I/O unit address for use during IPL operations.

Load Channel Address Bits: These four signal lines (+ parity) result from one of the three rotary-type LOAD UNIT switches on the SC. This hexadecimal character selects one of the 11 possible channels on the 9020 system.

Load Storage Select Bit: These four signal lines (+ parity) result from a rotary MAIN STORAGE SELECT switch on the SC. This hexadecimal character represents a main SE to be selected during IPL or manual operations.

Activate: This signal causes the SCON bits at the selected CE to be set according to the setting of the control CE switches on the SC.

Control CE: These four signal lines (+ parity) result from individual switches on the SC and allow manual setting of the SCON bits in the configuration register. Actual setting of the SCON field occurs with depression of the ACTIVE key on the SC.

All Stopped: This signal causes all CEs to enter the stopped state. The SYSTEM INTERLOCK switch must be turned on to activate the all-stop key, but the setting of the select CE switch does not affect this operation.

CE-CC INTERFACING

The CE-CC interface may be considered as divided into two sections:

1. CE control and indication.
2. RCU configuration and monitoring.

The greater portion of the interface is concerned with the first of these; that is, with sending indication (status) signals from the CE to the CC and control signals from the CC to the CE. The CE cannot initiate data transfer directly to the CC via this portion of the interface. Subsystem configuration indications, for example, which are under program control, are initiated by the CE and handled as a normal I/O operation via the IOCE.

Except for the line, 'subsystem load', this first portion of the interface is like the interface between a CE and the system console in a CCC system. The remaining portion is concerned with configuration, control, and monitoring of the two Reconfiguration Control Units (RCUs) in the CC. Specifically, this portion of the interface consists of the 'configuration data bus', 'reconfigure select/response' lines, and the 'system reset' lines from the CE to the RCUs, together with the 'element check' lines from the RCUs to the CE.

CE to CC Interface

Many of the indications from the CE to the CC represent the current status of the CE. The most important dynamic indications include the state of the CE, logic checks, current instruction address, and manual and wait status. In addition to lines indicating the status of the CE, there are lines concerned with configuration control for the RCUs. These include the 'configuration data bus', the 'reconfiguration select/response' lines, and the 'system reset' lines. The 'system reset' lines from CE to CC are strictly for resetting the RCUs. The CE to CC lines and buses are described below.

Data Indicator Lines: These 36 multiple-driver simplex lines provide for the display of four bytes of data from a main or local store location. They originate in the CEs at the T-Register.

Load Indicator: This multiple-driver simplex line originates in one or more CEs and turns on a common indicator. The indicator is lit by the respective CE from the time an IPL operation starts until it is complete.

Invalid Selection: This multiple-driver simplex line originates in one or more CEs and turns on a common indicator. The selected CE lights the indicator whenever an invalid or illegal storage address is specified by the operator during a manual operation.

Configuration Data Bus: This bus is an 11-bit distributed simplex bus which conveys the 'configuration mask' for the RCUs to the CC during execution of the SCON instruction by the CE.

System Reset (1, 2): These two distributed simplex lines provide a reset to the RCUs when activated by the CE.

Instruction Counter (IC) Indicators: These 27 simplex lines provide the CC with data for the display of the current instruction address at the CE. One set of these lines is indicated at the CC for each CE.

Manual Indicator: These simplex lines originate in each CE and terminate in a unique indicator at the CC to indicate when the associated CE is in the stopped state.

Wait Indicator: These simplex lines originate in each CE and terminate in a unique indicator at the CC to indicate when the wait bit in the associated CE's PSW is set to 1.

State Indicators: These four simplex lines originate in each CE and terminate in unique indicators at the CC. Only one of these four signals will always be active to reflect the present status of the originating CE.

Logic Check Indicator: These simplex lines originate in each CE and terminate in a unique indicator at the CC to indicate that the CE has detected one of its own logic check conditions.

Power Check: This signal line indicates that the temperature in the CE has drifted to within approximately 10 percent of the shutdown tolerance. The signal also indicates the loss of voltage or a normal power-off (but not an element master power-off) condition.

Battery: This signal indicates that the CE has switched to battery power.

Reconfiguration Select/Response: These two multiplex lines are used by the CE to signal the two RCUs to gate data from the configuration data bus into the CCRs.

Note: This same line is used by the RCU to respond to the SCON and may also be considered part of the CC to CE interface. Note also that there are physically three 'reconfiguration select/response' lines because the same circuitry is used for the RCU SCON bus in the CE as would be used for the three PAMs in a CCC system. Only two of the three are used by the RCUs.

CC to CE Interface

All CC controls to the CEs are gated by the system interlock switch, which requires a key operation to activate the logic. CC operations require that the CE TEST switch be off. In addition, all functions going to the CE (except for 'all stop') are further gated in the appropriate CE by a 'select CE' signal that originates from the select CE rotary switch on the system console.

One simplex ELC line originates from each RCU. The interface lines are described below.

Address Keys: These 24 signal lines (+3 parity) result from the 24 instruction address keys on the CC and provide addressing of any addressable local store or main storage location.

Data Keys: These 32 signal lines (+4 parity) result from the 32 storage data keys on the CC and provide manual data for storing into any addressable local store or main storage location.

CE Select: These four select lines result from the four-position rotary switch at the CC. Proper CE selection is under switch card control in the receiving CEs. This signal provides the necessary gating in the CEs for all manual operations (except 'all stop') issued from the configuration console. The SYSTEM INTERLOCK (key) switch must be turned on for this select switch to be enabled.

Stop: This signal places the selected CE in the stopped state without destroying its environmental status. The selected CE proceeds to the end of the instruction being executed at the time the 'stop' is initiated. If the current instruction causes a program interrupt, the change of program status words (PSW) will be accomplished before the stop. An I/O device will be allowed to complete its operation, although I/O or external interrupts will not be recognized.

Start: This signal starts the selected CE. If 'start' is issued after a manual stop, the CE continues as though no stop occurred.

Store Select Main: This signal results from the STORAGE switch being in the MAIN position and causes main storage addressing at the selected CE during either fetch or store manual operations. This signal line is not active with the storage select switch in the local store position and causes local storage addressing at the selected CE.

Display: This signal causes the selected CE to place the contents of a storage location, specified by the address keys and STORAGE switch, onto the data indicator lines to the CC.

Store: This signal causes the selected CE to store the contents of the CC storage data keys in the storage location specified by the 24 address keys and the STORAGE switch.

Set Instruction Counter (IC): This signal transfers the contents of the CC address keys to the instruction counter of the selected CE.

Interrupt: This signal results from depression of the interrupt key and causes a 'console interrupt' signal, which sets bit 25 in the PSW of the selected CE.

Address Compare Stop: This line conditions the selected CE so that any storage access to the address specified in the CC address keys causes the CE to enter the stopped state at the end of the instruction that made the memory reference.

Address Compare Loop: This line causes the selected CE to loop between the address set in the CC address keys and the address set in the storage data keys. When the selected CE makes a storage access to the address specified in the address keys, an unconditional branch is made at the end of the instruction to the address specified in the storage data keys. Programming can establish a loop condition between the two sets of keys.

Rate: This signal operates with the selected CE and the START and STOP keys on the CC. With this signal, each depression of the START key causes one complete instruction to be executed. Any machine instruction can be executed in this mode.

Load: This signal line initiates an IPL in the selected CE.

Subsystem Load: This line allows a subsystem IPL to be initiated from the CC.

Load Unit Address Bits: These eight signal lines (+ parity) result from two of the three rotary-type load unit switches on the CC. These two hexadecimal characters provide an I/O unit address for use during IPL operations.

Load Channel Address Bits: These four signal lines (+ parity) result from one of the three rotary-type load unit switches on the CC. This hexadecimal character selects one of the 11 possible channels on the 9020 system.

Load Storage Select Bit: These four signal lines (+ parity) result from a rotary main storage select switch on the CC. This hexadecimal character represents a main storage element to be selected during IPL or manual operations.

Activate: This signal causes the SCON bits at the selected CE to be set according to the setting of the control CE switches on the CC.

Control CE: These four signal lines (+ parity) result from individual switches on the configuration console and allow manual setting of the SCON bits in the configuration register. Actual setting of the SCON field occurs with depression of the ACTIVE pushbutton on the CC.

All Stopped: This signal causes all CEs to enter the stopped state. The SYSTEM INTERLOCK switch must be turned on to activate the all-stop key, but the setting of the select CE switch does not affect this operation.

Element Check: This simplex line is used to set a bit in the diagnose accessible register (DAR) in the CE when one or more of the following conditions exist at the RCU:

1. Power failure.
2. Parity check in the CCR.
3. Out of tolerance check (OTC).

An interruption is taken by the CE if the associated RCU bit is not masked off in the DAR mask and bit 7 of the PSW is not masked off. One ELC line originates in each RCU.

SECTION 1. INTRODUCTION

The appendix describes the operation of the 1052 adapter contained in the computing elements of the 9020E system. This adapter is not provided for computing elements of the 9020D system.

The 1052 adapter connects an I/O printer keyboard to a multiplexer channel in an IOCE. The printer and keyboard are mechanically and electrically independent, although they are both housed under the same cover.

The I/O printer keyboard provides two basic functions: facilities for manually entering data and facilities for printing data; both are under control of a monitor program.

When attached to a multiplexer channel operating in Multiplex mode, the adapter receives one eight-bit byte of data from, or transmits one to, the channel. The adapter then releases the channel to service other I/O devices. This byte mode of operation continues until the end of data transmission. With this mode of operation, the adapter releases the channel within the microsecond range for each byte of data.

CHANNEL INTERFACE SIGNAL SEQUENCE

One of the most important assumptions of this appendix is the reader's prior knowledge of the signal sequence of the channels to which the 1052 adapter can be attached. For review, the following material is presented at this time.

Initial Selection

Figure C-1 illustrates the sequence of interface signals for a complete write operation on a multiplexer channel. The channel itself begins the initial selection sequence. The channel raises the 'address out' and 'hold out' lines to all control units connected to the interface. The 'select out' line is raised to the control unit with the highest priority. The address byte designating a particular control unit is placed on the 'bus out' lines at the same time. The first control unit compares the address byte against its own address. If the two do not match, the control unit propagates 'select out' to the control unit with the next-lower priority. This attempt at address matching continues at each control unit in turn until the control unit whose address is on the 'bus out' lines is reached.

When the designated control unit makes the address match, the control unit raises the 'operational in' line to the channel. The channel recognizes the 'operational in' and drops the 'address out' line. When the control unit recognizes that the 'address out' line has been dropped, it raises the 'address in' line to the channel, and places its own address on the 'bus in' lines.

The channel accepts the address byte and the 'address in' tag line, and in response raises the 'command out' line and drops the 'select out' and 'hold out' lines. The channel also places a command byte on the 'bus out' lines at this time. For the sake of illustration, we are considering the command byte to indicate a Write-Inhibit Carrier Return (ICR) command.

The control unit reacts to the 'command out' line from the channel by dropping the 'address in' line, and the channel then drops 'command out' in response to the dropped 'address in' line. Now, the control unit raises the 'status in' line to the channel, indicating to the channel that the control unit has a status byte to send. The channel responds to the 'status in' line with 'service out', indicating the acceptance of the status byte. When the 'service out' line comes up, the control unit drops the 'operational in' line and the 'status in' line, and the channel in turn drops the 'service out' line. The initial selection sequence is complete.

Data Service

The sequence of signals during the data transfer is begun by the control unit raising the 'request in' line to the channel. The control unit raises 'request in' automatically for the first data byte following the initial selection sequence. The channel replies to the 'request in' with the 'hold out' and 'select out' lines. The 'address out' line is not raised by the channel at this time, because this is not the initial selection sequence. Channel has already established the fact (during initial selection) that it is in communication with this particular control unit. Therefore, as soon as the control unit recognizes the 'hold out' and 'select out' lines, it raises the 'operational in' line to the channel and, then, raises the 'address in' line to the channel and places its own address on the 'bus in' lines.

The channel responds with 'command out' at this time. During the initial selection sequence, the channel placed a command byte on the 'bus out' lines at the same time that it raised the 'command out' line. During the data transfer sequence, however, the bits on the 'bus out' lines are all 0's (with the exception of the parity bit). The 'command out' line rising at this time during the data transfer sequence, with the command byte bits set to 0's, indicates "proceed" to the control unit.

The control unit, in response to the 'command out' line from the channel, drops the 'address in' line, and the channel in turn drops the 'command out' line. The control unit reacts to the dropping of the 'command out' line by raising the 'service in' line. This indicates to the channel that the control unit is now ready for a data byte. The channel places the data byte on the 'bus out' lines and raises the 'service out' line to the control unit. The control unit receives both the data byte and the 'service out' line and, in turn, drops the 'service in' and 'operational in' lines. At this point, the data transfer sequence is complete. When the 1052 has finished printing the character, the control unit raises the 'request in' line to begin the next data transfer sequence to get the next character from the channel.

End Sequence

The data transfer sequences continue until all characters to be sent by the channel have been received by the control unit. When the control unit raises the 'request in' line, requesting another character, and when all of the characters have been sent, the channel begins the ending sequence. The sequences of the 'request in', 'select out', and 'hold out' lines are as previously described. The control unit again brings up 'operational in' in response to the 'hold out' and 'select out' lines and sends the 'address in' line and the address byte in to the channel. Channel again responds with 'command out', indicating proceed, and the control unit replies with 'service in', indicating that the control unit is ready for the data byte. The channel, however, replies with 'command out' instead of 'service out'. 'Command out' coming up in response to the 'service in' line indicates "stop" to the control unit.

When the control unit recognizes the stop indication, it drops the 'service in' line and indicates to the channel that it is going to send status to the channel by raising the 'status in' line. If the channel is able to accept the status byte at this time, it indicates this by raising the 'service out' line to the control unit. When the control unit receives the 'service out' reply from the channel in response to the 'status in' line, the operation is complete, and the 'operational in' line is dropped. The I/O control unit is finished with the full write operation and is free to be used for another operation by the channel.

The Write ICR command was considered here because it is a typical sequence. All other signal sequences may be treated as variations of these basic initial selection, data transfer, and ending sequences.

INTERFACE LINES

- Channel interface
- 1052 interface

The 1052 adapter has two sets of interface lines: those connecting to the channel and those connecting to the 1052. The interface lines between the channel and the 1052 adapter are:

From Channel to Adapter		From Adapter to Channel
	Bus Lines	
Bus Out Bit P		Bus In Bit P
Bus Out Bit 0		Bus In Bit 0
Bus Out Bit 1		Bus In Bit 1
Bus Out Bit 2		Bus In Bit 2
Bus Out Bit 3		Bus In Bit 3
Bus Out Bit 4		Bus In Bit 4
Bus Out Bit 5		Bus In Bit 5
Bus Out Bit 6		Bus In Bit 6
Bus Out Bit 7		Bus In Bit 7
	Tag Lines	
Address Out		Address In
Command Out		Status In
Service Out		Service In
	Scan Control Lines	
Select Out		Select In
Hold Out		Request In
Suppress Out		
	Interlock Lines	
Operational Out		Operational In
Clock Out		

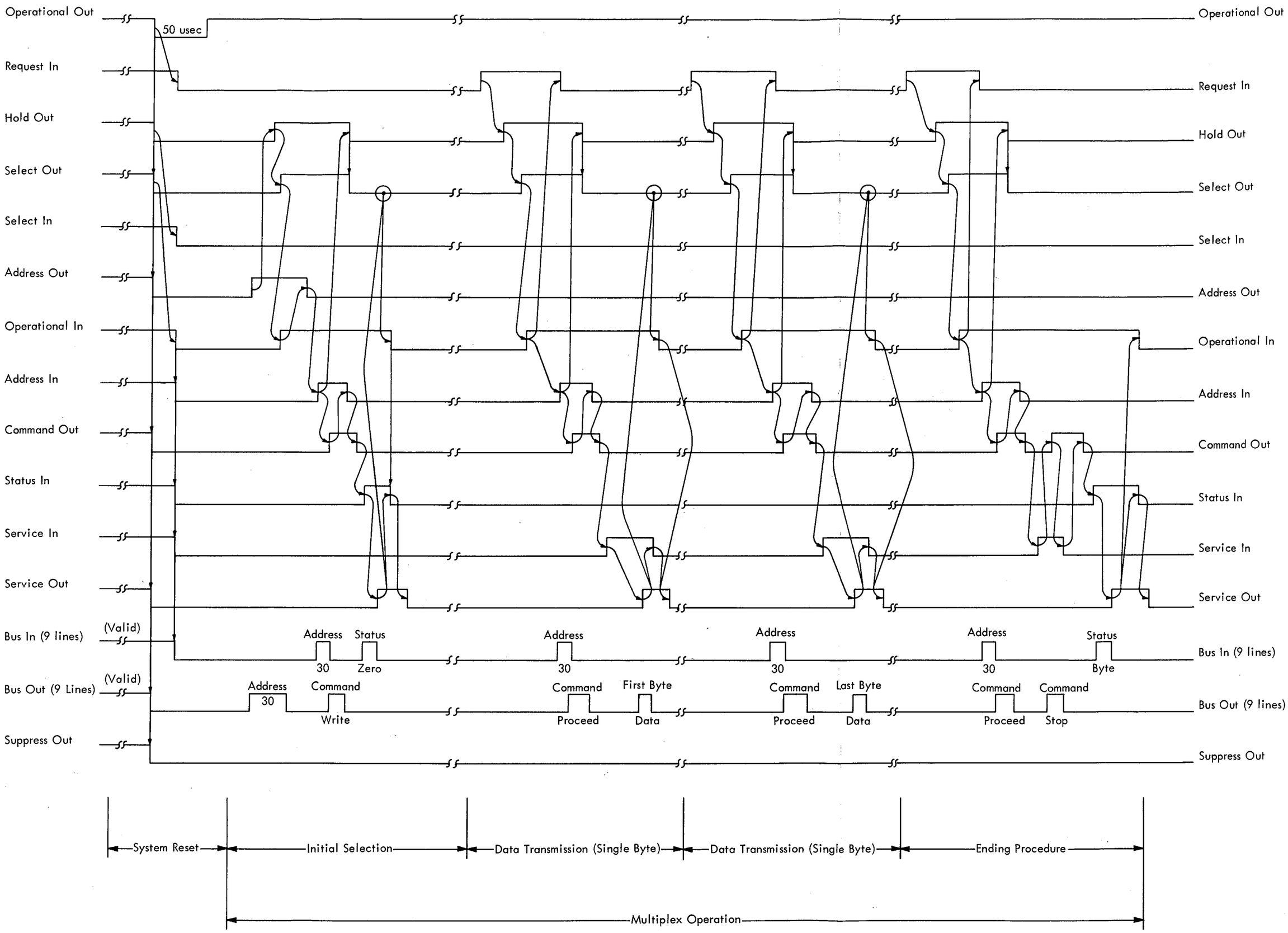


Figure C-1. Channel Interface Signal Sequence

The bus lines carry the data, sense, status, and address bytes between the channel and the 1052 adapter. The tag lines identify the bytes on the bus lines. The scan control lines control the sequence of priority of the various control units on the channel. The interlock lines provide the controls for connecting to, and disconnecting from, the channel.

The interface lines between the adapter and the 1052 are:

From Adapter to 1052	From 1052 to Adapter
T1	Keyboard BCD 1
T2	Keyboard BCD 2
R1	Keyboard BCD 4
R2	Keyboard BCD 8
R2A	Keyboard BCD A
R5	Keyboard BCD B
Aux (Check)	Keyboard BCD C
Space	Keyboard Strobe
Backspace	End of Forms Contact
Tab	RH Margin
Line Feed	1052 Busy
Carrier Return-Line Feed	1052 Not Busy
Upper Case Shift	Request PB N/O, N/C
Lower Case Shift	Ready PB
Lock Keyboard	Not Ready PB
CE (Test) Panel Indicators (20 lines)	Cancel PB
	Enter PB
	CE (Test) Mode
	CE (Test) Read
	CE (Test) Continuous Write
	Power On (+48V)

The first six lines to the 1052 energize the tilt/rotate magnets. Each of these magnets energizes the cycle clutch magnet. The seventh line picks the check magnet to energize the cycle clutch magnet for the rotate +5, tilt 3 character (none of the tilt or rotate magnets are picked for this character). The next seven lines energize the function magnets, and the last line operates the keyboard restore magnets to lock the keys. The 20 CE (test) panel indicators are shown in Figure C-13.

The first seven lines from the 1052 to the control unit are from the keyboard bail contacts, providing the BCD coding for each key. The eighth line provides an indication to the control unit that a BCD bit configuration has been set up in the bail contacts and that these contacts are to be sampled. The 'end of forms contact' line signals the control unit when the 1052 has run out of paper.

The tenth line from the 1052 to the adapter signals the adapter when the 1052 carrier has reached the end of the writing line. The adapter then causes the carrier to return to the left-hand margin. The 'carrier in motion' line indicates to the adapter that the carrier is in motion as the result of a carrier return or horizontal tab function.

The 'carrier in motion' line is deactivated when the carrier stops moving. The 'not cam cycle' line indicates to the adapter the failure of the printer to take a mechanical cycle (should it fail) when the printer is directed to print a character or to perform one of the following functions: up shift, down shift, space, backspace, or tabulate.

The next five lines are activated by the READY, NOT READY, and REQUEST pushbuttons (mounted on the cover, to the right of the carrier RETURN key) and by the CANCEL and ENTER pushbuttons (located at the left side of the keyboard). The CE (Test) panel switches (Figure C-13) provide the CE 'test', 'mode', 'read', and 'continuous write' signals. The 'power on' (+48V) signal is active if the keyboard-printer has +48V available from the supplying element (the CE).

COMMANDS

- Seven valid commands: Test I/O, Sense, Control No-Op, Control Alarm (No-Op), Write-ACR, Write-ICR, Read.

The 1052 adapter decodes commands from the channel and indicates the acceptance or rejection of the command to the channel. If the command is accepted by the adapter and requires a printer output (write), the adapter requests data from the channel. The data is received as a byte consisting of eight bits plus a parity bit. This eight-bit byte is translated into a tilt/rotate code, to pick the proper tilt/rotate magnets in the printer for a printable character, or into a function code to pick the proper function magnet in the printer.

If the command is accepted by the adapter and requires the operator to enter data (read), the operator keys in the desired character or printer function. The adapter receives the data from the keyboard one byte at a time, each byte consisting of six bits plus a parity bit. This data is translated into an eight-bit-plus-parity byte for subsequent transfer to the channel. The adapter requests service from the channel and places the character on the bus-in lines. When the channel accepts the data byte, the character is printed on the printer, or the printer function is performed. No commands (other than Read and Write) accepted by the adapter require the direct participation of the I/O device.

On the multiplexer channel, Read, Write, and Sense require an initial selection sequence; then the adapter disconnects from the channel. Each byte of data transmitted or received by the adapter requires the adapter to first transmit its address and then request service from the channel. Upon acknowledgement of the request by the channel, the adapter again disconnects from the channel. This procedure continues until data transmission is ended.

Read

Upon acceptance of the Read command, the keyboard is unlocked, and the proceed light is lit. The proceed light indicates that the operator may key in a character. When the operator keys in a character, it is parity checked (six bits plus parity) from the keyboard, translated to an eight-bit byte, and set into the data register. The adapter then requests service from the channel. When the channel accepts the data byte, the proceed light is extinguished, the keyboard is locked, the character is translated to a tilt/rotate or function code, and the printer either prints the character or performs the function. The keyboard is unlocked, and the proceed light is again turned on. This procedure continues until a termination is indicated.

A Read command may be terminated in one of three ways: byte count equals 0 at the channel, an 'enter' signal given by the keyboard operator, or a 'cancel' signal given by the keyboard operator. The enter 'signal' is issued as a result of the operator's depressing the ENTER pushbutton. The 'enter' signal is used to indicate the end of a block of data. The 'cancel' signal is issued by the operator's depressing the CANCEL pushbutton. The 'cancel' signal is used to indicate an operator error in the message; in this case, the entire message must be re-entered.

When the adapter detects any of the above terminating signals, it locks the keyboard (which stays locked until a new Read command is issued), turns off the proceed light, initiates a carrier-return operation, and sends a Channel-End status to the channel. The unit exception status bit is sent with the Channel End if the termination was accomplished by the 'cancel' signal. When the carrier is returned to the left-hand margin, the adapter requests service from the channel to present a status byte consisting of Device End. The unit check status bit sent with Channel End or Device End indicates one or more of the following conditions: BCD parity error from the keyboard, no mechanical cycle of the printer, or an error between the BCD keyboard output and the output of the printer translator.

Write

Two Write commands are available: Write-Auto Carrier Return, and Write-Inhibit Carrier Return. Write-Auto Carrier Return automatically returns the carrier to the left margin at the end of the write operation. Write-Inhibit Carrier Return does not cause the carrier to return at the end of the write operation.

Write-ACR

Upon acceptance of the Write command, the adapter disconnects from the channel. It then requests service from the channel for each data byte and disconnects after each data byte. When the 'stop' signal is presented by the channel, the adapter sends Channel-End status and disconnects from the channel. When the carrier is returned to the left margin, the adapter signals for a device-end interrupt. Device-End status is sent when the channel responds.

Write-ICR

This command proceeds the same as Write-ACR, except that upon receipt of the 'stop' signal the adapter transmits both Channel-End and Device-End in one status byte.

Test I/O

If the Test I/O command reaches the adapter and no outstanding status bits are present, a zero status byte is returned to the channel.

If status information is pending, all status bits present (except busy, when caused by Channel End, Device End, Attention, or status stacked) are transmitted to the channel.

Control No-Op

The Control No-Op command is a control immediate. Channel-End and Device-End status are transmitted together in the initial selection sequence.

Control Alarm (No-Op)

The Control Alarm command is a control immediate. This command resets the sense byte and causes a general selective initial reset. Channel End and Device End are transmitted during the initial selection status.

Sense

The Sense command transmits one eight-bit byte plus parity to the channel. This byte consists of the contents of the 'sense' triggers. After this data transfer, a final status byte consisting of Channel End and Device End is sent to the channel. The adapter does not disconnect from a multiplexer channel between transmission of the sense byte and final status (forced Burst mode operation).

1052 ADAPTER PRIORITY

- Determined at time of installation
- Pluggable

The priority of the 1052 adapter is not necessarily dictated by the physical placement of the control unit on the I/O

interface. The adapter can be plugged to receive either 'select out' or 'select in' as the scanning line. The priority of the scanning line is set at the time of installation. The adapter provides the option of connecting its selection logic in series on either the 'select out' or the 'select in' line. Ascending-order priority from the channel can be established on the 'select out' line, and the remaining control units can maintain a descending-order priority from the channel on the 'select in' line. Plugging location for the select-out bypass card is 02ED1-B6 (refer to ALD ZP080). See Figure C-2.

DATA FLOW

- Diagram 6-28, FEMDM illustrates both data flow and control.
- Data flow path from data register to printer is the same for both read and write.

Controls

The interface controls govern the sequence of signals the 1052 adapter sends to the channel on the inbound 'tag' and 'selection control' lines. The interface controls are acted upon by both the active and inactive states of the outbound 'tag' and 'selection control' lines. They are also acted upon by the command decoder, the command register, and the address compare circuits.

The address and command bytes are received from the channel on the 'bus out' lines. The address byte is compared with a fixed address plugged within the 1052 adapter. The command decoder decodes the command byte to determine whether the command is one of the seven acceptable to the adapter and, if so, to determine which of the seven commands it is.

Write Data Path

Data bytes are received on the 'bus out' lines during a write operation. Each data byte is checked for parity when it is taken from the 'bus out' lines, and the parity bit is then dropped. The eight-bit byte is stored in the data register by the data register controls. The contents of the data register are then analyzed by the function decoder to determine whether the character in the data register is a function character or a printable character.

If the data register contains a printable character, that character is translated [by the eight-bit to tilt/rotate (printer) translator] from the eight-bit code to the tilt/rotate code.

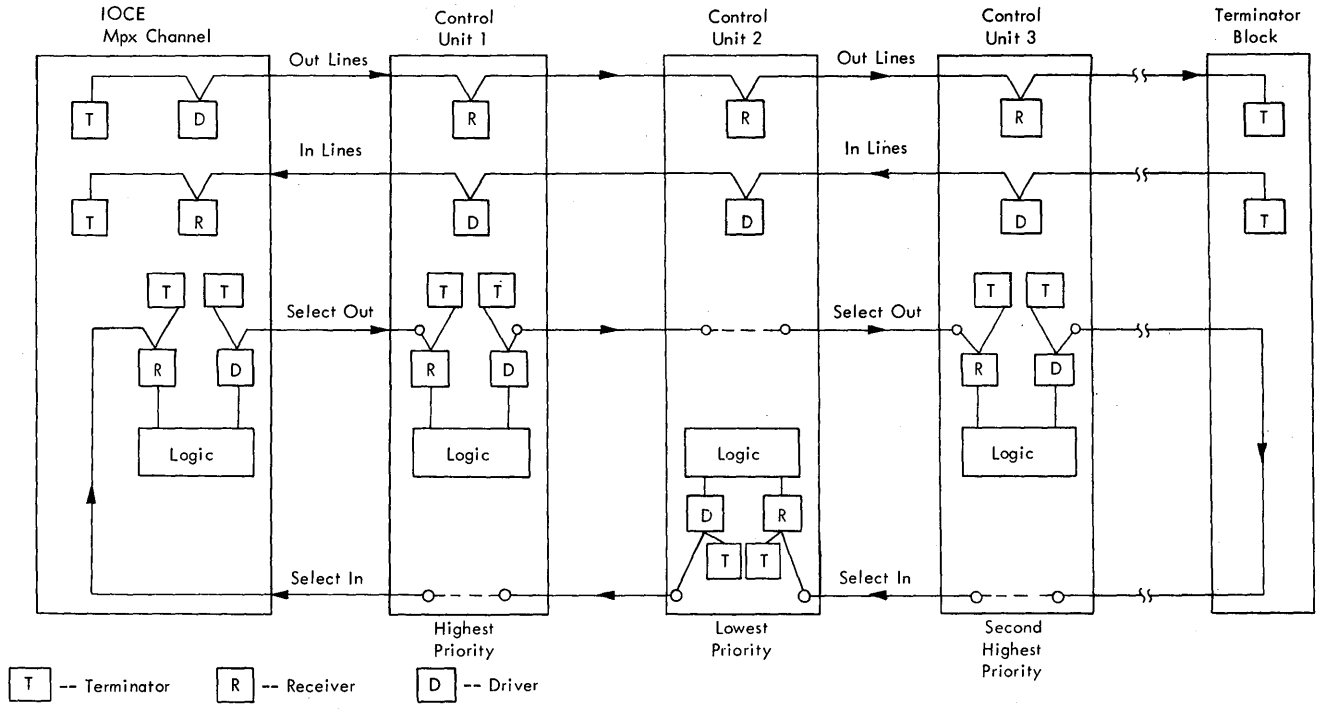


Figure C-2. I/O Interface Interconnections

Read Data Path

The keyboard generates data and function bytes in a six-bit-plus-parity code during a read operation. The output of the keyboard is powered, parity checked, and fed to the BCD-to-8-bit (keyboard) translator. The keyboard translator translates the BCD code to the eight-bit code, and the data register controls store the eight-bit character in the data register. (Figure C-3 illustrates the translation of the keyboard code and the corresponding printer output.) At this point, the interface controls request service from the channel.

When the channel responds, the data byte is gated to the 'bus in' lines through the parity generator. The parity generator inserts P-bits as required to make the byte on the 'bus in' lines odd parity. The byte is sent to the printer at the same time it is sent to the 'bus in' lines. The character path to the printer from the data register is the same for both read and write.

Address-In, Sense, and Status Bytes

The address generator is gated to the 'bus in' lines during those I/O interface signal sequences requiring the adapter to send its own address into the channel. Because the parity bit is plugged along with the rest of the bits in the address generator, the output of the address generator is not passed through the parity generator.

The output of the sense-bit latches is gated to the 'bus in' lines during the execution of a Sense command. This byte is passed through the parity generator because the bit structure of the byte varies, and the parity bit must be inserted as needed. The output of the 'status' triggers is gated to the 'bus in' lines during an initial selection sequence and during an ending sequence. Correct parity must also be generated for the status byte because the structure of the status byte is also variable.

6-Bit Code (PT and T) BA8421	Lower Case		Upper Case	
	Keyboard Print Function	8-Bit Code (EBCDIC) 01234567	Keyboard Print Function	8-Bit Code (EBCDIC) 01234567
000001	1	11110001	=	01111110
000010	2	11110010	<	01001100
000011	3	11110011	;	01111111
000100	4	11110100	:	01110100
000101	5	11110101	%	01101100
000110	6	11110110	'	01111101
000111	7	11110111	>	01101111
001000	8	11111000	*	01011100
001001	9	11111001	(	01001101
001010	0	11110000	)	01011101
001011	#	01111011	"	01001111
010000	@	01111100	¢	01101110
010001	/	01100001	?	01100011
010010	s	10100010	S	11100010
010011	t	10100011	T	11100011
010100	u	10100100	U	11100100
010101	v	10100101	V	11100101
010110	w	10100110	W	11100110
010111	x	10100111	X	11100111
011000	y	10101000	Y	11101000
011001	z	10101001	Z	11101001
011011	,	01101011	I	01001111
100000	-	01100000	-	01110010
100001	j	10010001	J	11010001
100010	k	10010010	K	11010010
100011	l	10010011	L	11010011
100100	m	10010100	M	11010100
100101	n	10010101	N	11010101
100110	o	10010110	O	11010110
100111	p	10010111	P	11010111
101000	q	10011000	Q	11011000
101001	r	10011001	R	11011001
101011	$	01011011	!	01011010
110000	&	01010000	+	01001110
110001	a	10000001	A	11000001
110010	b	10000010	B	11000010
110011	c	10000011	C	11000011
110100	d	10000100	D	11000100
110101	e	10000101	E	11000101
110110	f	10000110	F	11000110
110111	g	10000111	G	11000111
111000	h	10001000	H	11001000
111001	i	10001001	I	11001001
111011		01001011	¬	01011111
000000	Space	01000000		
001110	Up-Shift	——		
011101	LF	00100101	(Line Feed)	
101101	CRLF	00010101	(Carrage Return Line Feed)	
101110	BS	00010110	(Backspace)	
111101	TAB	00000101	(Horizontal Tab)	
111110	Downshift			

Figure C-3. 1052 Keyboard Code Translation and Printer Output

- Data register
- Read/write clock
- Printer translator
- Keyboard translator
- Shift controls
- Function decoder
- 1052 Printer-Keyboard

DATA REGISTER

- Used for Read command
- Used for both Write commands

The data register comprises eight latches, one for each of the bit lines 0 through 7. Each latch has three possible set and reset lines. Figure C-4 illustrates the input and output controls for the data register.

Write Operation

One group of set lines for the eight latches comes from the 'bus out' lines from the channel. These lines are ANDed with SS 1, not CE (test) mode, and not busy or stop to set the data byte on the 'bus out' lines into the data register. The function decode circuits activate the 'function' line at SS 2 time if the character in the data register is a function character. SS 2, Write command, and not busy, sets cycle time.

Cycle time gates either the function decoder output (function character in the data register) to energize a function magnet in the printer, or the 8-bit-to-tilt/rotate (printer) translator output (printable character in the data

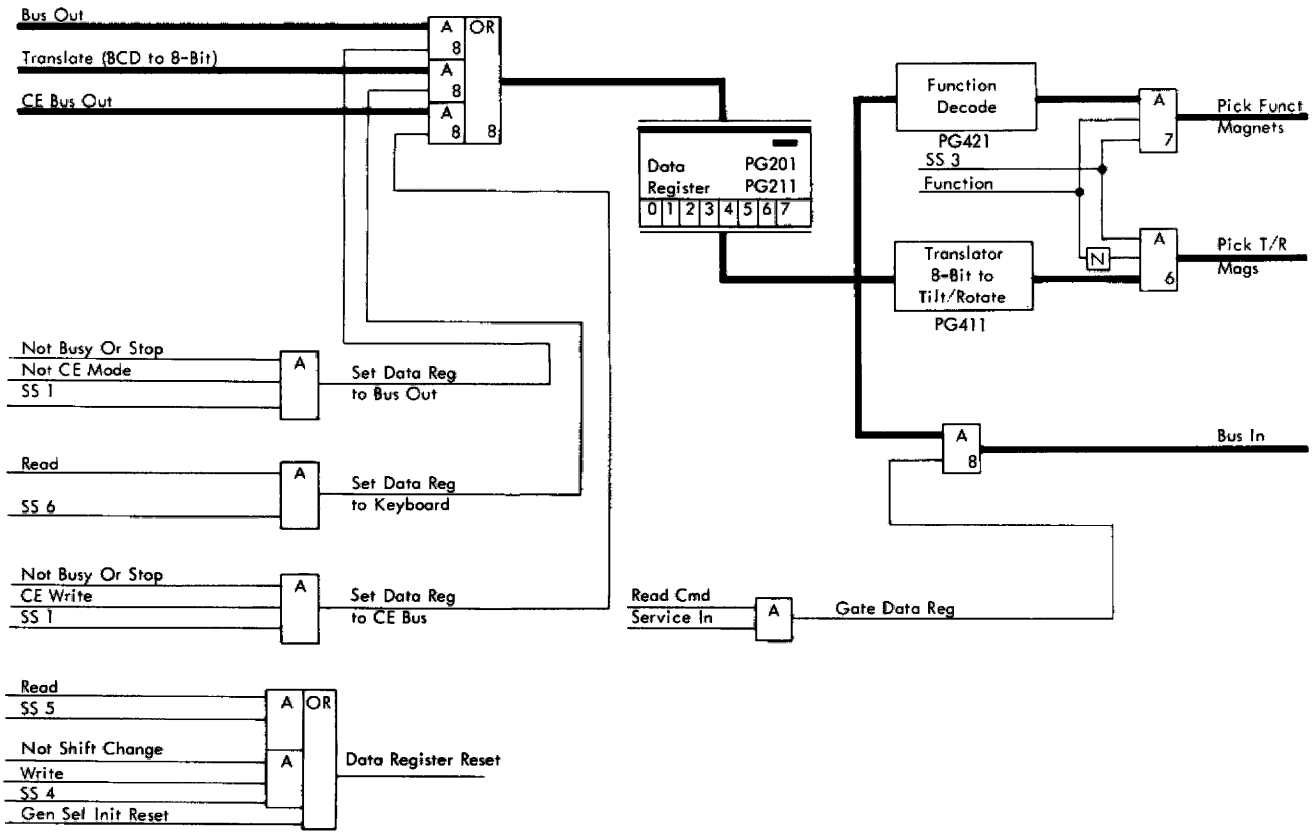


Figure C-4. Data Register, Input and Output Controls

register) to energize the tilt/rotate and cycle clutch magnets in the printer. Cycle time also sets the 'printer busy' latch when SS 2 times out.

SS 4 ANDs with 'write' and 'not shift change' to reset the data register. The 'not shift change' line indicates that the character in the data register did not require case-shifting the printer. If the data register had contained an uppercase character and the printer had been in lowercase, or vice versa, the character would have been held in the data register while the printer was shifted. The clock would have run through SS 4 during the shifting, but the data register would not have been reset, the reset having been blocked by the 'not shift change' line being inactive. When the shift is complete, the clock is again started at SS 1. The data register cannot be set again because the 'printer busy' latch is on when the clock starts again at SS 1. When the clock runs through SS 4 the second time, the data register is reset.

Read Operation

A second group of set lines for the data register latches comes from the BCD-to-8-bit (keyboard) translator. The read/write clock is started at SS 5 for a read operation, and SS 5 ANDs with read to reset the data register. SS 6 ANDs with read to gate the translated keyboard output into the data register. SS 7 requests service from the channel by raising the 'request in' line to the channel, indicating that the 1052 adapter has a data byte ready for transfer to the channel.

When the channel responds and connections are established between the channel and the adapter, the 'service in' line (ANDed with Read command) gates the output of the data register to the 'bus in' lines. The channel responds to 'service in' with 'service out', and the adapter ANDs 'service in' with 'service out' to start the read/write clock at SS 3. From this point, the data byte proceeds to the printer exactly as described for the write operation. The data byte has now been placed on the 'bus in' lines and has also been printed.

READ/WRITE CLOCK

- Used for Read and Write commands.
- Used to return carrier, space, and shift.
- Start and stop points vary with operation.

The read/write clock is used for three of the seven valid commands accepted by the control unit: Read, Write-Auto Carrier Return, and Write-Inhibit Carrier Return. The clock is stepped sequentially from SS 1 through SS 7 for both of the Write commands.

During the data transfer sequence of a Write command, the adapter activates the 'service response' line (Figure C-5) by ANDing the 'service in' and 'service out' tag lines. Service response then starts the clock by firing SS 1. SS 1 gates the data byte on the 'bus out' lines into the data register (Figure C-3).

SS 2 fires when SS 1 times out. SS 2 controls the 'lowercase/uppercase' latch and the 'shift change' latch (Figure C-8). The outputs of SS 2 and SS 3 are overlapped, but the duration of SS 2 is only 200 ns while the duration of SS 3 is 40 ms. The effective duration of SS 3 then is 39.8 ms. When SS 2 has finished timing out, the 'printer busy' latch is set by SS 3 ANDed with not SS 2.

SS 3 is ANDed with the outputs of the function decoder and the printer translator to pick the function or the tilt/rotate magnets in the 1052 (Figures C-6 and C-9).

SS 4 fires when SS 3 times out. For a Read command the SS 4 output resets the 'shift change' latch if it has been set. It also sets the 'printer busy' latch for the Read command. For the Write command, if the 'shift change' latch has not been set, SS 4 resets the data register and fires SS 5.

For the Read command, SS 5 is fired by the 'read' line ANDed with the keyboard strobe. The read/write clock is started at SS 5 for the Read command. When SS 5 times out, it fires SS 6 if the carrier is not being returned to the left-hand margin. If the carrier is in the process of being returned, the 'carrier in motion' latch is on, and SS 6 cannot be fired until it is turned off.

Note the 'carrier in motion' latch (Figure C-5). The latch is turned on when the normally open carrier return interlock contacts make. The latch is turned off when these contacts open. When the latch turns off, the 30-ms SS is fired; this blocks the AND function that sets the 'carrier in motion' latch. This means that the latch cannot be turned on again for 30 ms after it has been turned off. This is a bounce-protection device. If the carrier return interlock contacts bounce closed after opening, the 'carrier in motion' latch will not be set again.

SS 6 is used to gate the output of the BCD-to-8-bit (keyboard) translator to the data register for the Read command. SS 6 also turns on the 'service request' latch, to indicate to the channel that the adapter is ready for another data byte (Write command) or ready to send another data byte to the channel (Read command).

SS 7 fires when SS 6 times out. SS 7 resets the 'shift change' latch for the Write command. It also fires SS 3 during the Read command if the 'shift change' latch has been set. The clock outputs are related in detail to the various operations in Section 3 of this appendix.

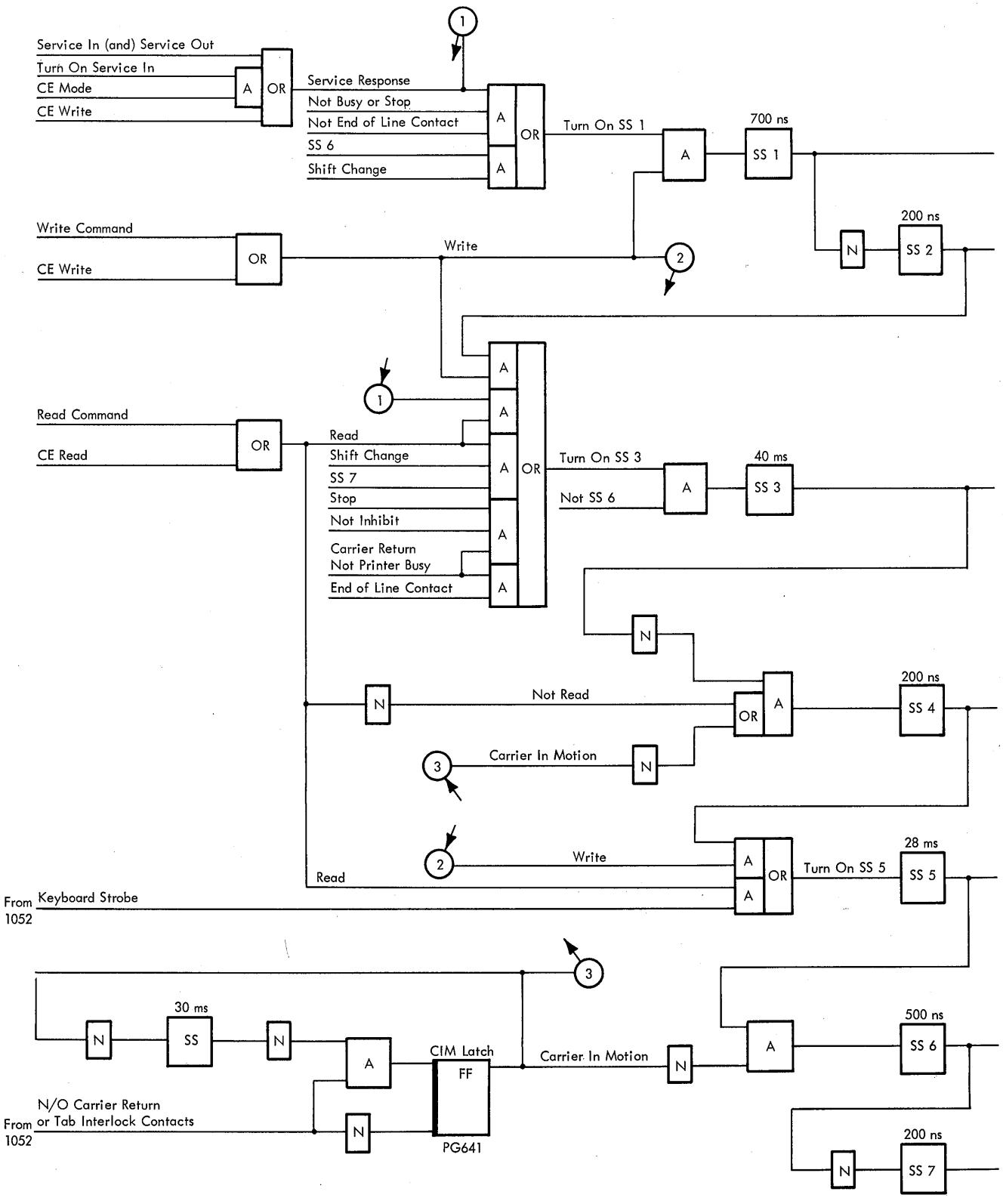


Figure C-5. Read/Write Clock

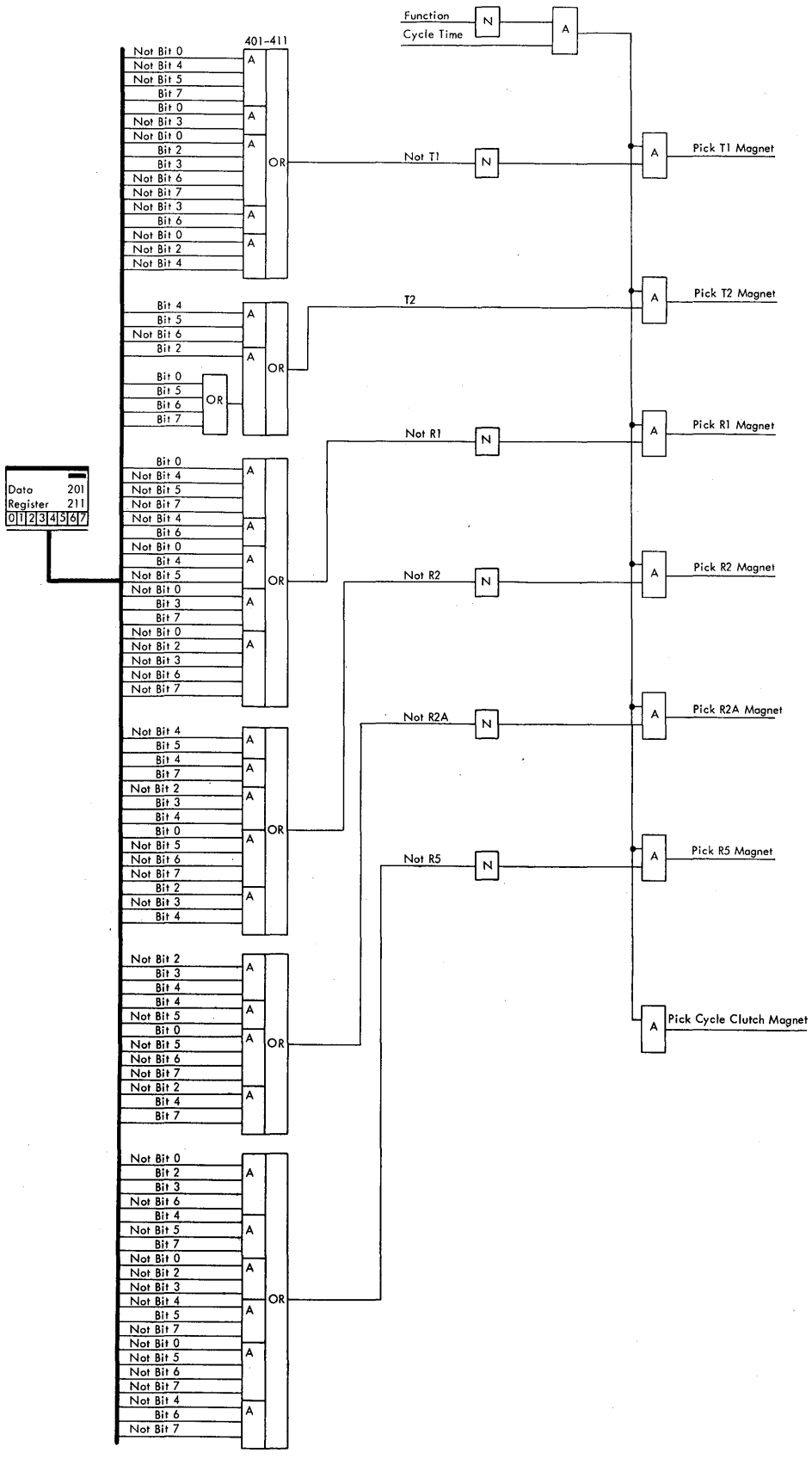


Figure C-6. Printer Translator (Translate 8-Bit to Tilt/Rotate)

PRINTER TRANSLATOR

- Translates eight-bit code to tilt/rotate code.

Figure C-6 illustrates the 8-bit-to-tilt/rotate translator. The series of ANDs and ORs connected to the data register output decode the eight-bit byte stored in the data register into the corresponding tilt/rotate code. Note that all translating AND/OR functions except T2 use negative logic; that is, making the OR prevents the energizing of the corresponding tilt/rotate magnet. T2, the exception, is picked when the OR is made.

The "even" function at the bottom of Figure C-6 energizes the aux (auxiliary, or check) magnet if an even number of tilt/rotate magnets is energized. Since one is an odd number of magnets, none is an even number; therefore, if no magnets are energized, the even function picks the aux magnet. The result of picking the aux magnet alone is a tilt motion of +3 and a rotate motion of +5. If two, four, or six tilt/rotate magnets are picked, picking the aux magnet has no effect.

The output of this translator is gated with SS 3 and 'not function'. If the byte in the data register is a function character, the function decoder (Figure C-9) activates the function line at SS 1 time, when the data byte is set into the data register. Conversely, if the data byte in the data register is not a function character, the function line is inactive and allows SS 3 to gate the lines picking the tilt/rotate and aux magnets.

KEYBOARD TRANSLATOR

- Translates six-bit code to eight-bit code.

Figure C-7 illustrates the translation of the BCD output of the 1052 keyboard to the eight-bit code. Not all of the AND/OR logic is illustrated because all functions are similar. Note that four of the eight output lines are negative logic in that activating the output line prevents the corresponding bit from being set into the data register. The other four output lines are positive logic; activating these lines results in setting the corresponding bit into the data register.

Also note the 'lower/upper case' latch. The output of this latch is ANDed into the translator to control the translation of the BCD output of the keyboard. Each key results in the same BCD coding for both the upper and lower case. The translation to either uppercase or lowercase eight-bit code is controlled by the 'lower/upper case' latch. For example, the output of the letter key "B" is B A $\overline{8}$ $\overline{4}$ 2 $\overline{1}$ in the BCD code. This code is the same for both upper and lower case. When the 'lower/upper case' latch is on (lower case), the B A $\overline{8}$ $\overline{4}$ 2 $\overline{1}$ is translated to 0 $\overline{1}$ $\overline{2}$ $\overline{3}$ $\overline{4}$ $\overline{5}$ 6 $\overline{7}$. When the 'lower/upper case' latch is off (upper case) B A $\overline{8}$ $\overline{4}$ 2 $\overline{1}$ is translated to 0 1 $\overline{2}$ $\overline{3}$ $\overline{4}$ $\overline{5}$ 6 $\overline{7}$. The control of the 'lower/upper case' latch is described in the following paragraphs under "Shift Controls".

SHIFT CONTROLS

- Controls case hemisphere of typing element.

Figure C-8 illustrates the 'shift change' latch, the 'lower case/upper case' latch, and the circuits that decide whether the character in the data register is an uppercase character or a lowercase character. The only circuit that produces no interaction between the 'lower case/upper case' latch and the 'shift change' latch is the function that ANDs the 'carrier return' and 'stop' latches. This AND function activates the 'turn on lower case' line at the end of a read operation and at the end of a write-auto carrier-return operation, to insure that the typing element in the printer is returned to the lowercase hemisphere.

During a write operation, the 'lower case/upper case' latch and the 'shift change' latch are controlled by the output of the decision circuits at the top of Figure C-8. If the 'lower case/upper case' latch is on, indicating that the printer is in lowercase and the character in the data register is an uppercase character, the printer must be shifted to upper case before the character can be printed. The 'upper case character' line is activated by the uppercase character in the data register. 'Upper case character' is ANDed with 'lower case' latch on and SS 2 to turn the 'lower case/upper case' latch off and, also, to turn on the 'shift change' latch. The 'shift change' latch coming on causes the character to be held in the data register while the clock is run to shift the printer to the upper case. After the shift cycle is complete, the clock is again run to print the character.

Had the printer been in upper case, and had the character in the data register been a lowercase character, the printer would have been shifted to lower case before gating the output of the data register to the printer translator. If the printer had been in upper case, the 'lower case/upper case' latch would have been off, and SS 2 would have been ANDed with not upper case character and lower case latch off to turn both the 'shift change' latch and the 'lower case/upper case' latch on. Again, the 'shift change' latch would have caused the character in the data register to be held there while the clock was run to shift the printer to the lower case. After the shift cycle was completed, the clock would have been run to print the lowercase character.

The read operation is similar to the write operation. The major difference lies in the control of the two latches. When the shift key on the keyboard is operated, the keyboard sends the BCD bits $\overline{C}$ $\overline{B}$ $\overline{A}$ 8 4 2 $\overline{1}$ to the adapter. The keyboard strobe is also sent to the adapter. The

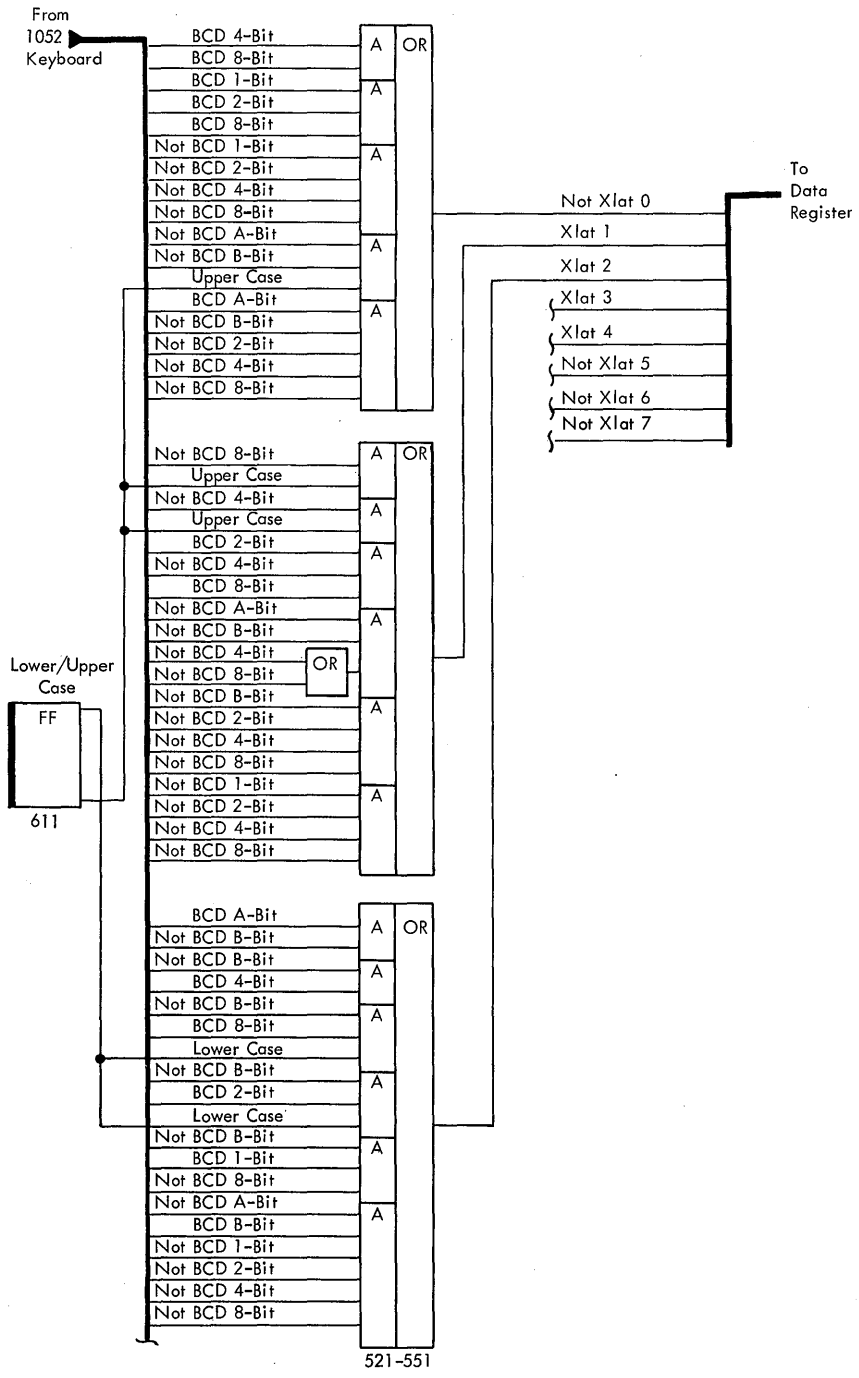


Figure C-7. Keyboard Translator

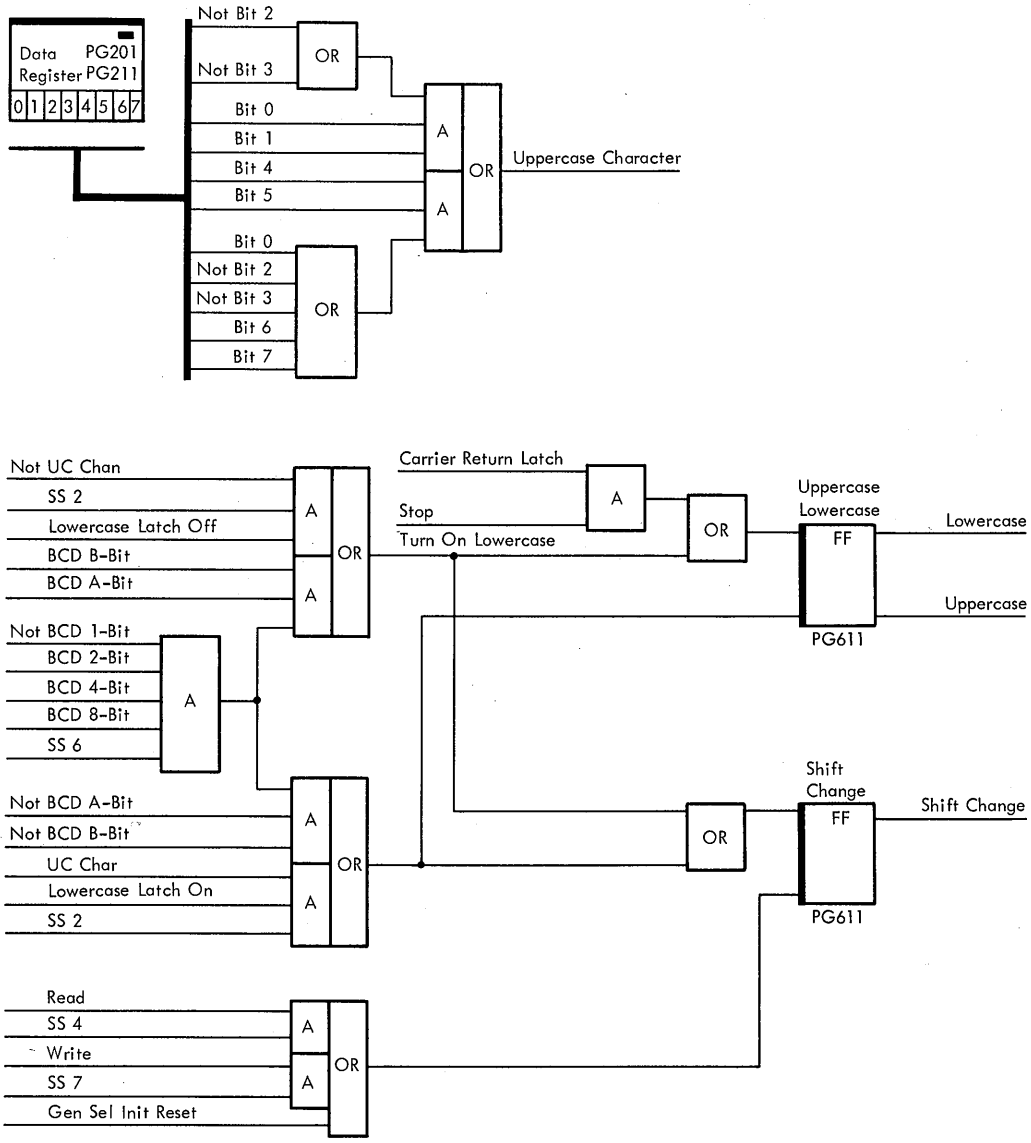


Figure C-8. Shift Controls

keyboard strobe starts the clock at SS 5, and the upshift BCD bit configuration is ANDed with SS 6 to turn off the 'lower case/upper case' latch and turn on the 'shift change' latch. The 'shift change' latch causes the read/write clock to run to shift the printer from lower case to upper case. The 'shift change' latch also blocks SS 7 from setting the 'service request' latch to raise request-in to the channel. No byte is sent to the channel for shifting.

When the shift key is released at the keyboard, a sequence similar to that described in the above paragraph occurs. The BCD bits differ in that instead of $\overline{C}\ \overline{B}\ \overline{A}\ 8\ 4\ 2\ \overline{1}$ (upshift) the keyboard sends $\overline{C}\ B\ A\ 8\ 4\ 2\ \overline{1}$ to the adapter along with the keyboard strobe. The 'lower case/upper case' latch is turned on along with the 'shift change' latch. The clock is started by read ANDed with keyboard strobe; the printer is shifted to the lower case, and the request-in line is blocked.

FUNCTION DECODER

- Determines whether the data register contains function of printable character.
- Operates printer function magnets.

Figure C-9 illustrates the function decode circuits. These circuits analyze the character in the data register to decide whether that character is to perform a function at the printer. Just as the tilt/rotate magnets are energized by

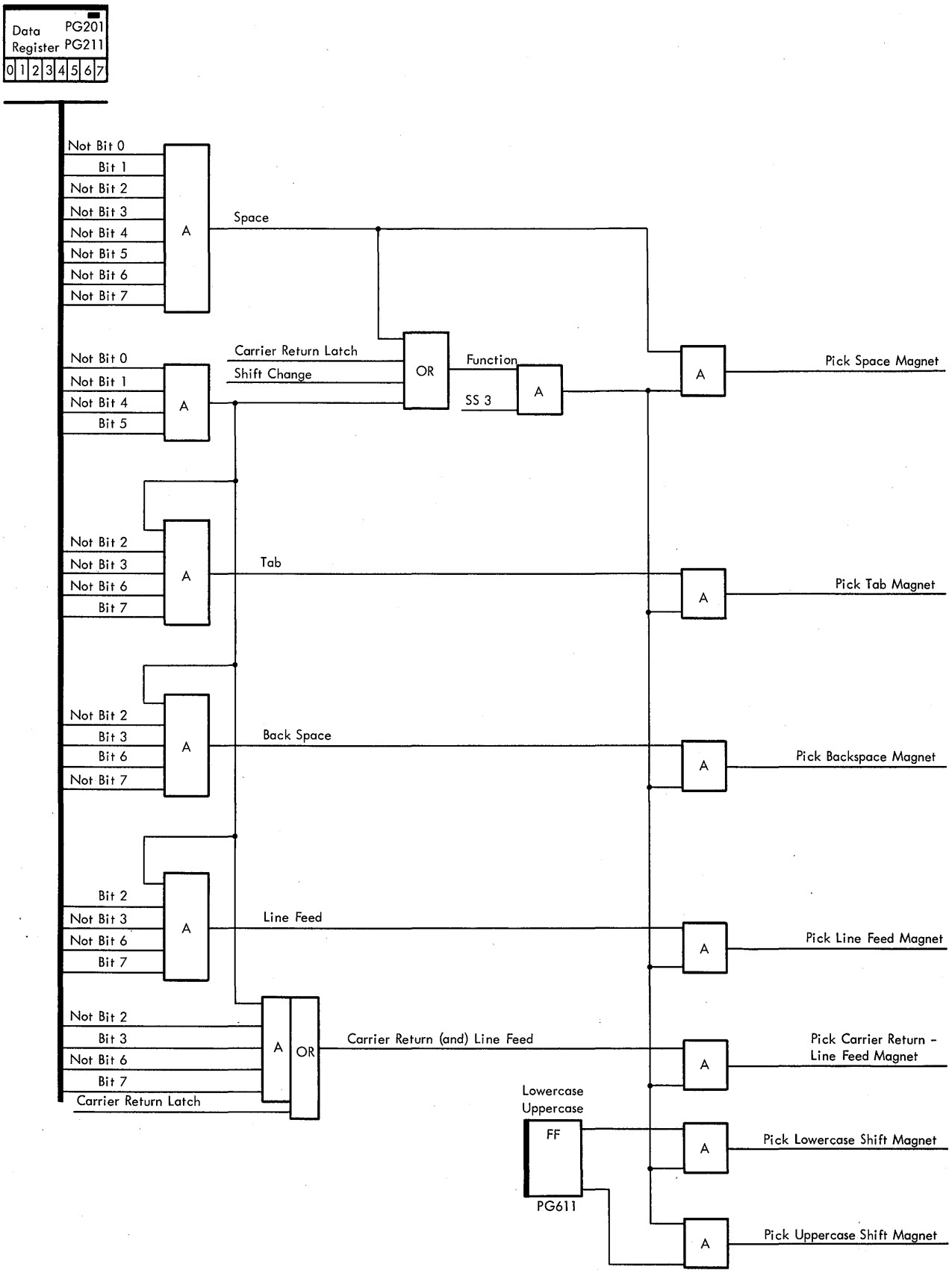


Figure C-9. Function Decoder

cycle time, so are the function magnets. The eight-bit bytes for the space and carrier return-line feed characters activate the 'function' line. This line is activated at SS 1 during a write operation when the data register is set with the character on the 'bus out' lines, or at SS 6 when the data register is set with the output of the keyboard translator for the read operation.

Note that the carrier return-line feed magnet may be operated by either a carrier return-line feed character in the data register or the 'carrier return' latch. Note also that the 'function' line is activated by either the 'shift change' latch, the 'carrier return' latch, or any function character in the data register.

The lowercase shift magnet and the uppercase shift magnet are operated from the 'lower/upper case' latch. Any time the case of the printer is to be changed, the 'shift change' latch is turned on, and the output of the 'shift change' latch activates the 'function' line. Cycle time then may be ANDed with the on or off output of the 'lower/upper case' latch to perform the shifting in the printer.

1052 PRINTER-KEYBOARD

- Selectric printer
- Keypunch keyboard

The IBM 1052 Printer-Keyboard is a page printer with printing mechanism similar to the IBM Selectric* typewriter. The 1052 keyboard is a modified IBM 024/026 keyboard housed in the same cover as the printing mechanism of the IBM Selectric typewriter. The printer and keyboard are electrically (not mechanically) connected by means of the 1052 adapter, which allows these two units to operate independently.

Printer

The 1052 printer is a self-contained package including the drive motor. It is designed for placement on a flat surface convenient to the operator. The printer is cable-connected to the 1052 adapter, which contains the 8-bit-to-tilt/rotate (printer) translator. The printer can accept data at a maximum rate of 15.5 characters per second from the channel. The printer has a removable print head, allowing the selection of different print arrangements. The arrangement of the characters on the print head is illustrated in Figure C-10.

The printer recognizes 44 printable characters in the Upshift mode, and 44 in the Downshift mode. The printer performs four functions: space, carrier return and line feed, upshift, and downshift.

A paper presence control constitutes part of the interlock circuitry that places the printer in a Ready status. As the trailing edge of the last form reaches a point about 2 inches from the printing line, the printer signals this condition to the adapter and reverts to a Not Ready status. Printing is not interrupted by this action.

The horizontal tab and carrier return-line feed interlocks interrupt the printing operation to allow enough time for the printer to complete these functions. More than one character time will normally be required. When the printer begins either a tab or a carrier return-line feed function, the 'carrier in motion' latch is set in the adapter. The 'carrier in motion' latch interlocks the adapter until the printer has completed the function. When the function is completed, the printer resets the 'carrier in motion' latch, and the adapter resumes its operation.

The basic printing operation is as follows: The tilt and rotate selector latches in the printer are controlled by the tilt and rotate (tilt/rotate) magnets in the printer. If the character in the data register in the adapter is a printable character, the corresponding 'tilt/rotate' lines in the adapter are activated to energize the tilt/rotate magnets in the printer. If the character in the data register is a function character, the corresponding printer 'function' line is activated to energize the function magnet in the printer.

Keyboard

The 1052 keyboard is an adaptation of the IBM 024/026 keyboard, with the numeric keys located in a fourth bank similar to a typewriter keyboard. The keyboard is mechanically independent of all other units and is connected electrically only to the 1052 adapter. Data may be sent from the keyboard to the channel and/or the printer only through the adapter.

The basic arrangement of the four-bank keyboard is shown in Figure C-11. There is a total of 53 function and character keys. Each key, except the ALTN CODING key, generates a BCD code for the character or function associated with that key. The two SHIFT keys and the shift LOCK generate the up shift code when depressed. The down shift code is generated when either SHIFT key is released. Output from the keyboard is the IBM paper tape and transmission code and is in odd parity. Figure C-12 illustrates the arrangement of the bail, latch, and keystem contacts in the keyboard.

*Trademark of IBM, Inc.

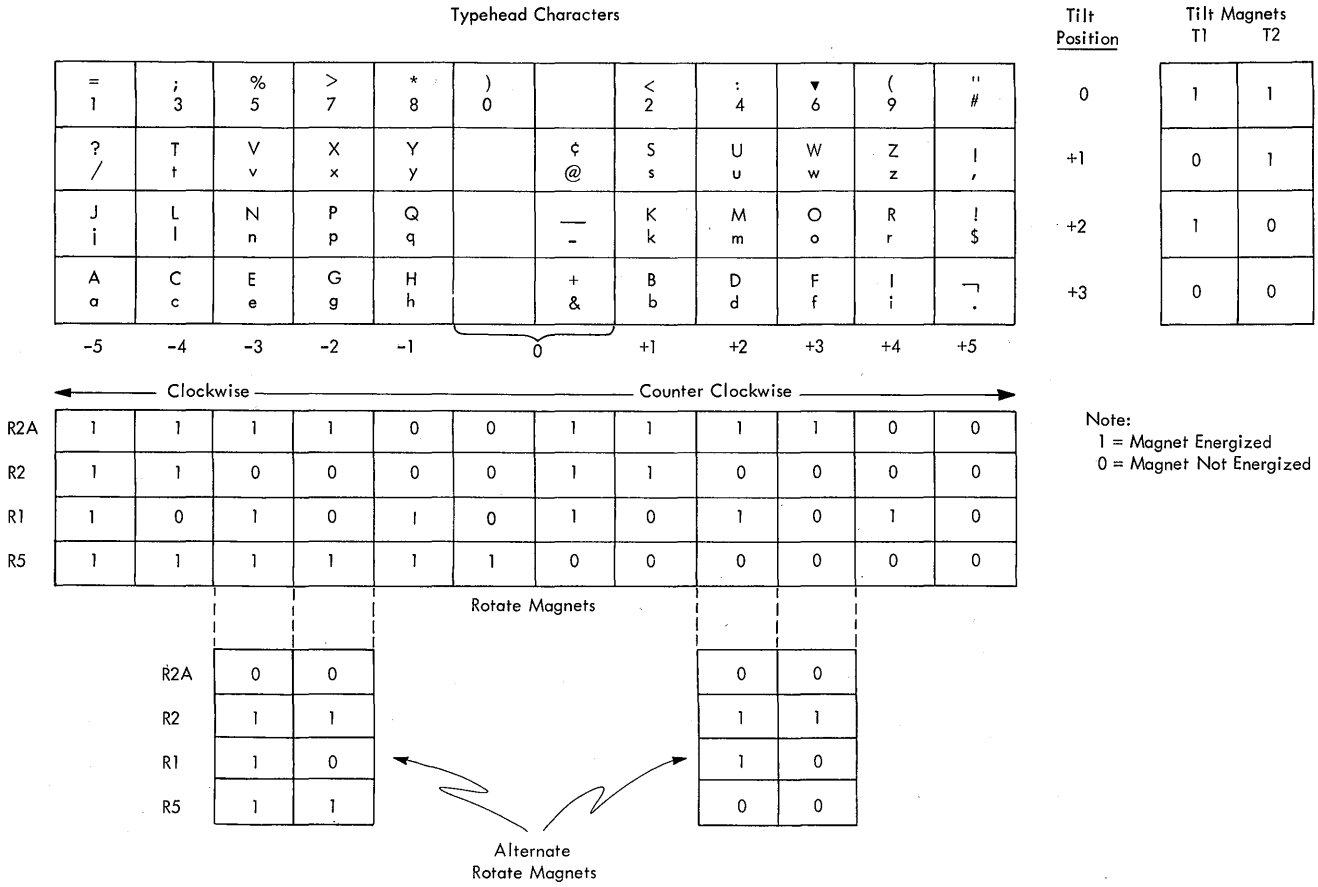


Figure C-10. 1052 Internal Tilt/Rotate Code

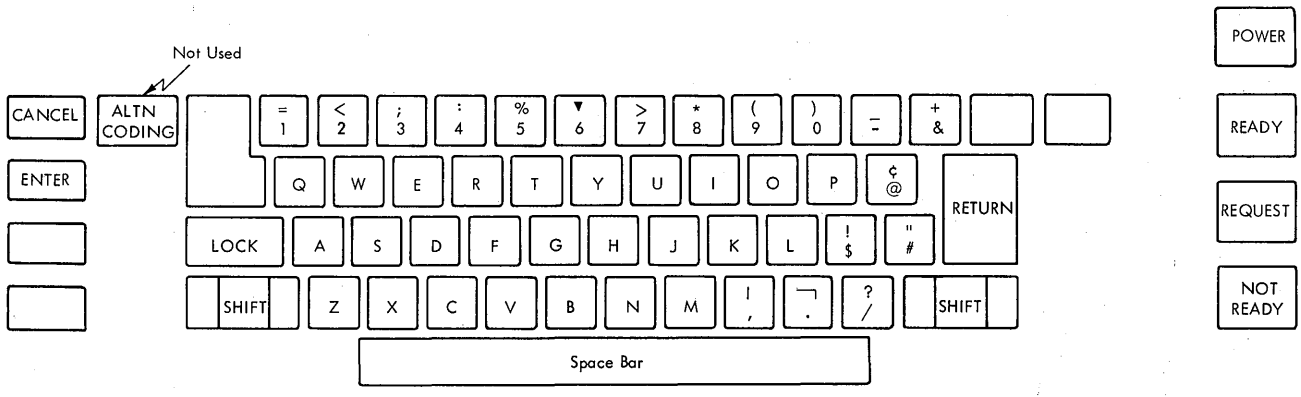
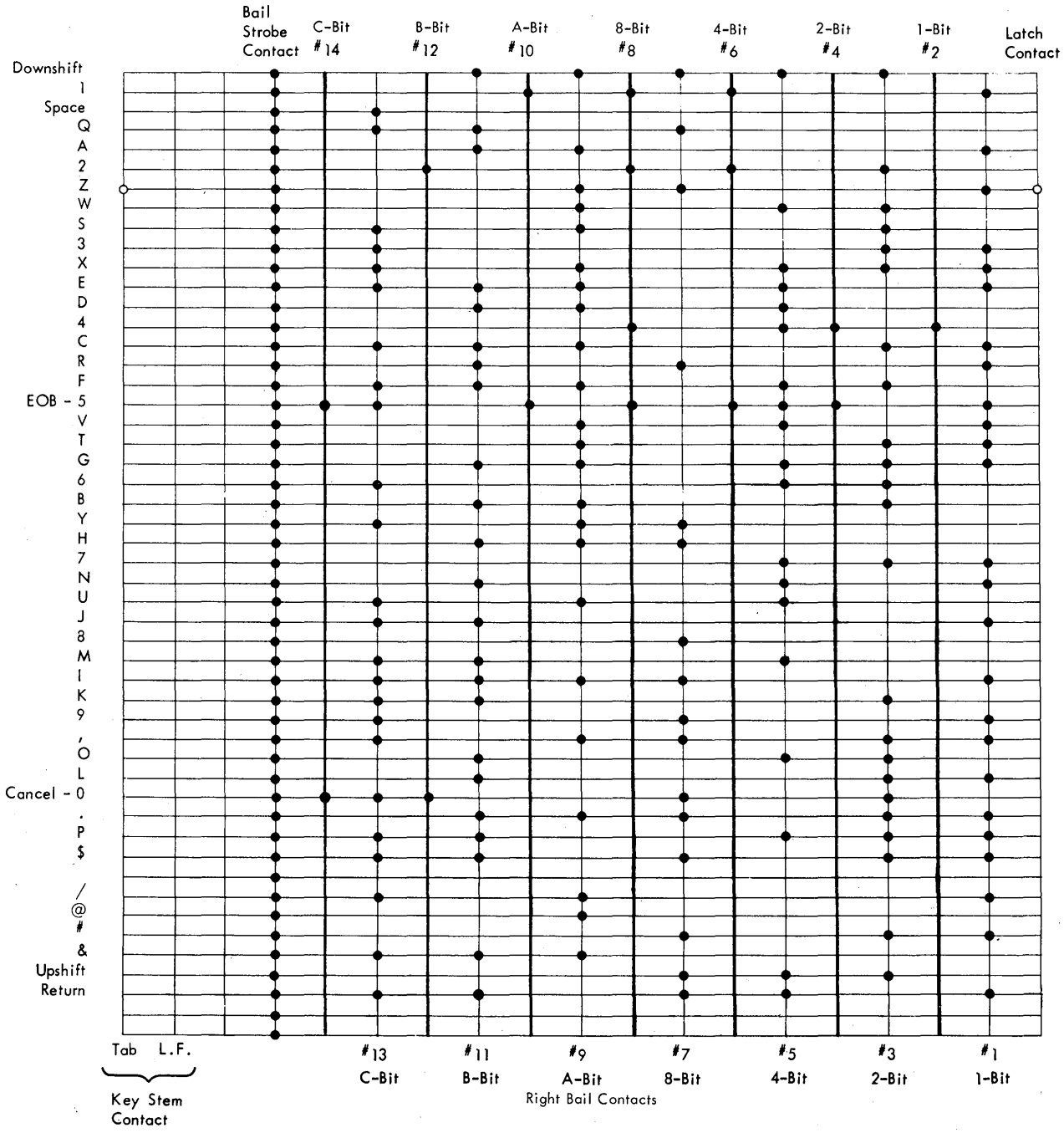


Figure C-11. 1052 Keyboard



Keyboard Common Contact Numbers 1 and 2. These contacts open when the keyboard-restore magnets are energized. Number 1 common contact disconnects the supply voltage from the bit lines and prevents sending any bits during a keyboard-restore operation.

Number 2 common contact opens the direct circuit to the restore magnets and puts a current-limiting resistor in series with the restore magnets. This permits locking the keyboard by continually energizing the restore magnets.

Left-Hand Latch Contact (Tab and Z). This latch contact closes whenever the Z or tab key is pressed. When the tab key is pressed, the tab keystem contact is closed and the B- and 4-bit keyboard lines are pulsed to complete the tab code.

Strobe Bail Contact. When any permutation bar drops, the strobe bail contact makes and starts the read/write clock in the adapter.

Right-Hand Latch Contact (Carrier Return and Line Feed, and Line Feed-CR/LF and LF). The contact closes whenever the return or line-feed key is pressed. When the return key is pressed, the right-hand latch contact closes and pulses the B-bit keyboard line to complete the CR/LF code. When the line-feed key is pressed, the right-hand latch contact closes and the line-feed keystem contact also transfers, and the A-bit keyboard line is pulsed to complete the line-feed code.

Figure C-12. 1052 Latch, Bail, and Keystem Contact Arrangements

KEYBOARD CONTROLS

- REQUEST pushbutton
- READY pushbutton
- NOT READY pushbutton
- ENTER pushbutton
- CANCEL pushbutton

REQUEST Pushbutton

The REQUEST pushbutton is mounted on the 1052 cover, to the right of the keyboard. When operated, it sets the 'store request' and the 'attention status' latches. (See Diagram 6-33, FEMDM.) The 'attention status' latch, in turn, activates the 'attention interrupt' line, and the 'attention interrupt' line activates the 'status conditions' line. (See Diagram 6-31, FEMDM.) The 'request in' line is raised to begin a status transfer sequence with the channel in which the status byte with 0-bit transferred to the channel. The 0-bit then indicates that the REQUEST pushbutton has been operated.

Note the store request latch in Diagram 6-33, FEMDM. The 'store request' latch and the unnamed latch to the left form a single reset-dominant-latch configuration. The 'store request' latch is reset by the 'reset attention stored' line, and the unnamed latch is reset by the 'inverted request pushbutton' line.

The 'store request' latch may be reset within nanoseconds after the REQUEST pushbutton is operated, and the operator may still be holding the pushbutton after the reset drops. When this condition occurs, the unnamed latch prevents the pushbutton from setting the latch again. The pushbutton must be released and depressed once more to set the 'store request' latch again.

READY and NOT READY Pushbuttons

The READY and NOT READY pushbuttons are mounted on the 1052 cover, to the right of the keyboard. The READY pushbutton sets the 'ready' latch. (See Diagram 6-33, FEMDM.) The NOT READY pushbutton resets the 'ready' latch. The 'ready' latch may also be reset by the end-of-forms contacts when the printer runs out of paper.

A 6-bit is set into the status byte by the 'unit check' latch when the 'ready' latch is reset. When the 'ready' latch is first set, it fires a singleshot to set the 'store device end' latch. (See Diagram 6-31, FEMDM.) The 'device end interrupt' line then requests a status transfer sequence from the channel to signal the channel that the 1052 is ready; no status bits are set.

ENTER Pushbutton

The ENTER pushbutton signals the adapter to terminate a read operation in the normal manner. When the operator is through keying data to the channel, he depresses the ENTER pushbutton. The 'enter' signal to the adapter sets the 'stop' and 'device end' triggers, initiating the ending sequence. (See Diagram 6-31, FEMDM.)

CANCEL Pushbutton

The CANCEL pushbutton notifies the system that a keying error has been made during a read operation. When the operator is keying data to the channel and makes an error, he may depress the CANCEL pushbutton to terminate the read operation. The 'cancel' signal sets the 'cancel' latch in the adapter. The 'cancel' latch sets the 'stop', 'channel end', and 'unit exception' latches. 'Channel end' initiates the ending sequence. Presenting 'unit exception' status to the CE causes the program to ignore the erroneous data and allows the operator to enter correct data.

POWER Indicator

The POWER indicator indicates that the printer-keyboard has +48V dc available from the supplying element (the CE).

CE (TEST) PANEL

A CE (test) panel is mounted in the vertical portion of the 1052 cover, just above the keyboard. This panel is illustrated in Figure C-13. There are two switches on the panel: CE MODE/ON LINE, and CONTIN WRITE/READ. The CONTIN WRITE/READ switch is inoperative when the CE MODE/ON LINE switch is set to the ON LINE position. When on-line, the 1052 is operated in conjunction with the channel and the 1052 adapter.

Switches

For diagnostic purposes, the 1052 and the adapter can both be taken off-line by setting the CE MODE/ON LINE switch to the CE MODE (Test) position. The adapter and the 1052 are then under control of the CONTIN WRITE/READ switch. When the CE MODE/ON LINE switch is set to the CE MODE and the CONTIN WRITE/READ switch is set to

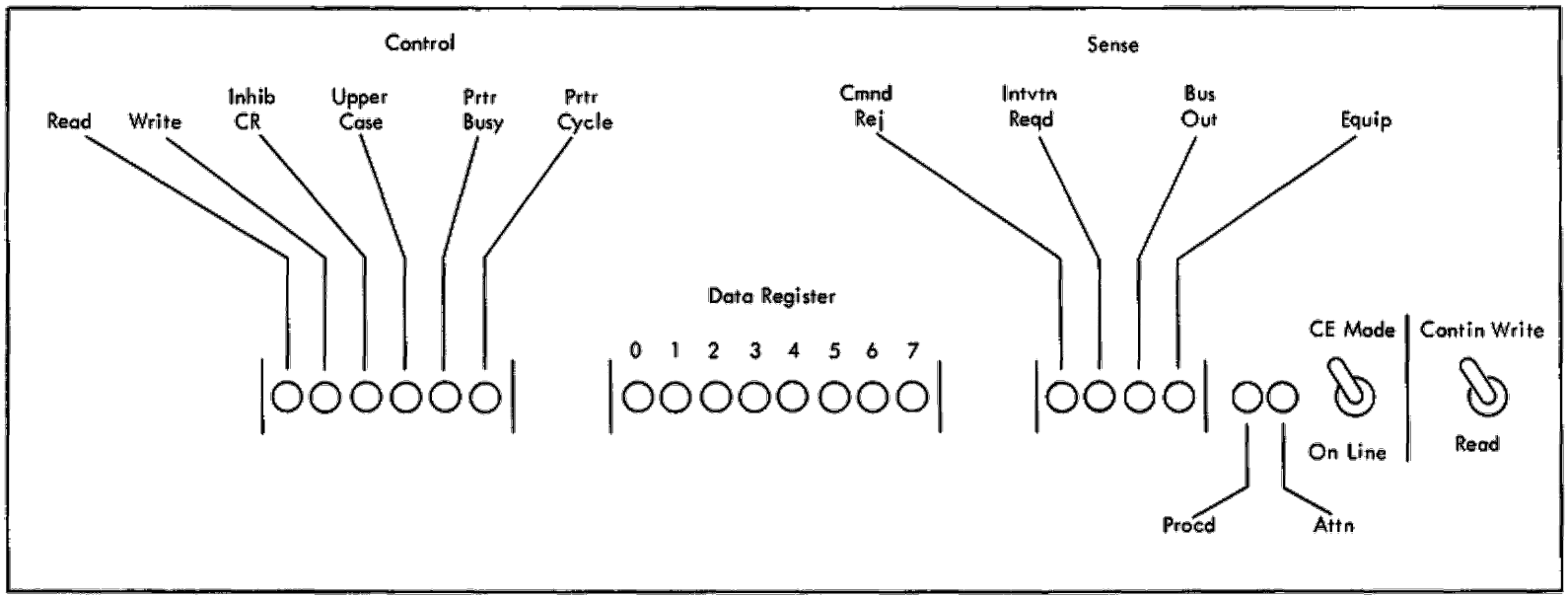


Figure C-13. 1052 CE (Test) Panel

CONTIN WRITE, the output of a character emitter in the adapter is gated to the data register. From the data register, the characters follow the normal data path for a write operation. The character emitter must be plugged by the maintenance personnel to emit two characters in the eight-bit code. Figure C-14 illustrates the operation of the emitter. Note, in the figure, that when the switches are set to CE MODE and CONTIN WRITE, the 'CE write' line is activated. The 'CE write' line starts the read/write clock with SS 1 (Figure C-5), gates the CE bus to the data register (Figure C-3), and also operates the binary trigger at SS 4 of each clock cycle.

The Continuous Write mode provides a semifixed means to check up to 90 percent of the printer controls within the adapter, check decoding of the EBCDIC code to appropriate printer functions and characters, and mechanically check the printer portion of the 1052. The Continuous Write mode allows the maintenance personnel to set up any two EBCDIC codes and perform the desired operations alternately by the printer. The plugging is performed on the pin side of SLT board X1. The board is factory-wired (yellow) for zero alternating with uppercase A.

SLT board X1 location for the 7201-02 CE is 02E D1.

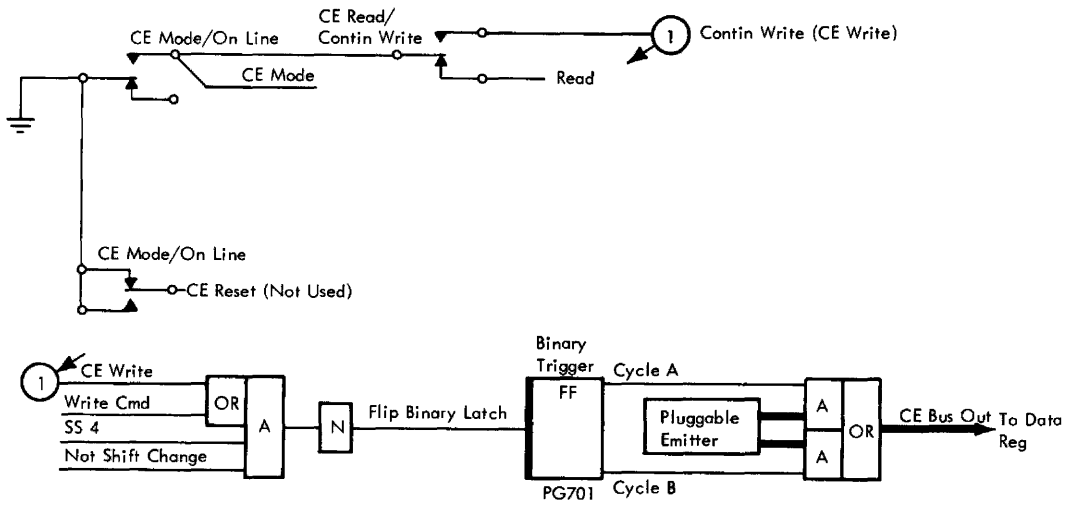
See Figure C-15 for plugging instructions and Figure C-16 for the EBCDIC code set.

CE test read permits keying in of all characters on the keyboard and performing the desired operation, i.e., printing or printer/function. All read controls can be checked, including PTTC/8 to EBCDIC translation and EBCDIC to tilt/rotate or printer function. The data register indicators contain the PTTC/8 to EBCDIC translation and remain set until a subsequent character is keyed.

Lights

The six CONTROL lights on the test panel indicate the controlled state of the 1052. The READ lamp is lit whenever the 'read command' latch in the adapter is on or when the switches on the panel are set to the CE MODE (test) and READ. The WRITE lamp is lit whenever the 'write command' latch in the adapter is on or when the switches on the panel are set to CE MODE (test) and CONTIN WRITE. The INHIB CR lamp is lit whenever the 'inhibit carrier return' latch is on in the adapter. The UPPER CASE lamp is lit whenever the 'lower/upper case' latch in the adapter is off, indicating that the printer and keyboard are in upper case. The PRTR BUSY lamp is lit whenever the 'printer busy' latch in the adapter is on, (the printer is taking a cycle). The PRTR CYCLE lamp is lit when the 'printer cycle' latch in the adapter is on. It indicates that the printer failed to take a mechanical cycle when signalled to print a character or perform a function (space, backspace, tab or case shift).

The DATA REGISTER lamps indicate the contents of the data register. Each lamp indicates the on-state of its associated 'data bit' latch. The SENSE lamps indicate the on-state of the 'sense bit ' latches. The PROCD (proceed) lamp is lit whenever the keyboard is unlocked to allow the keying of data or functions from the 1052 keyboard. The ATTN (attention) lamp is lit when the REQUEST push-button on the 1052 keyboard cover is operated. The lamp remains lit until the 'gate status in' line comes up. The lamp indicates the on-state of the 'store request' latch.



Note: 'CE' equals 'test'.

Figure C-14. CE Write

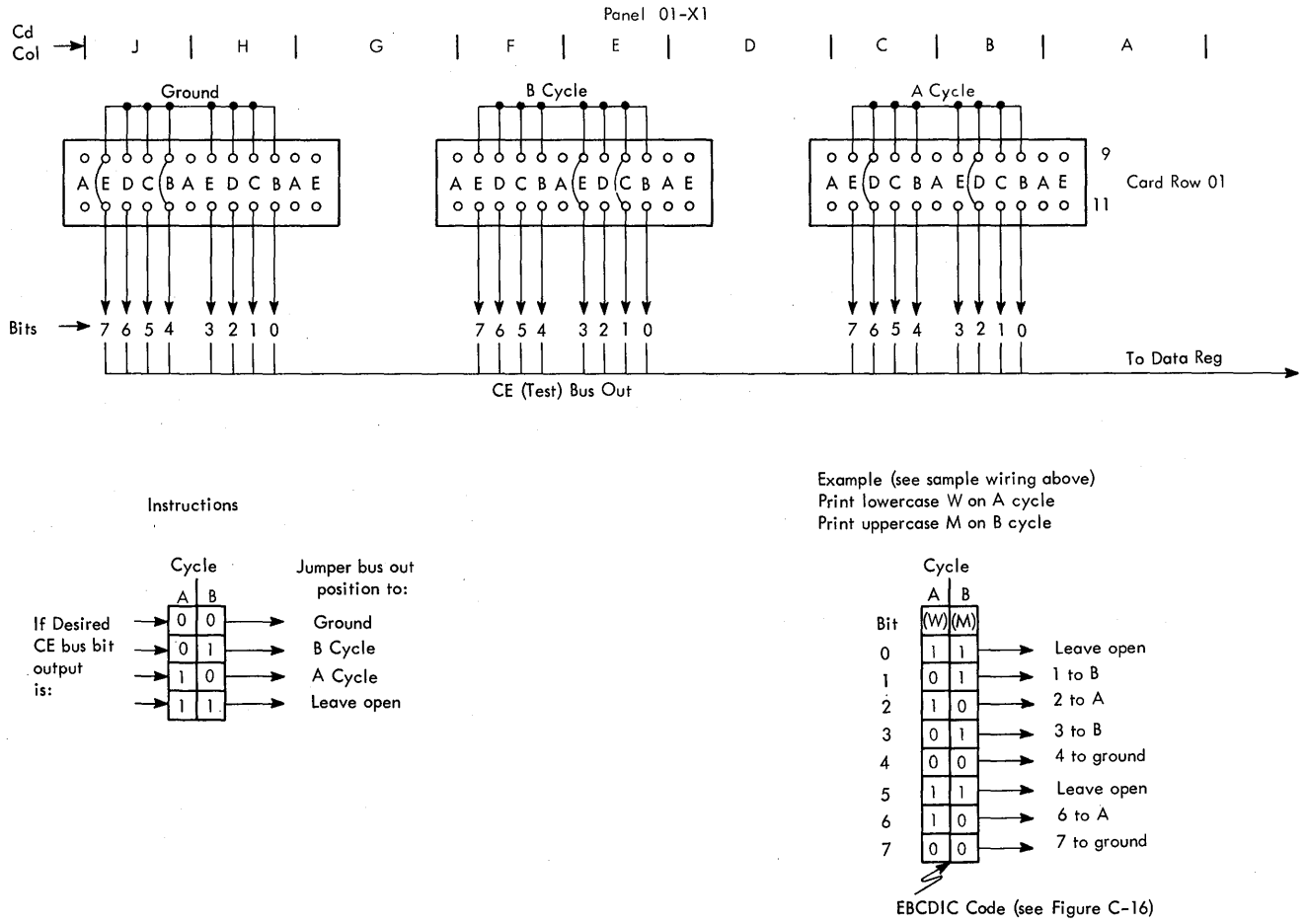


Figure C-15. CE (Test) Continuous Write Mode-Wiring Chart

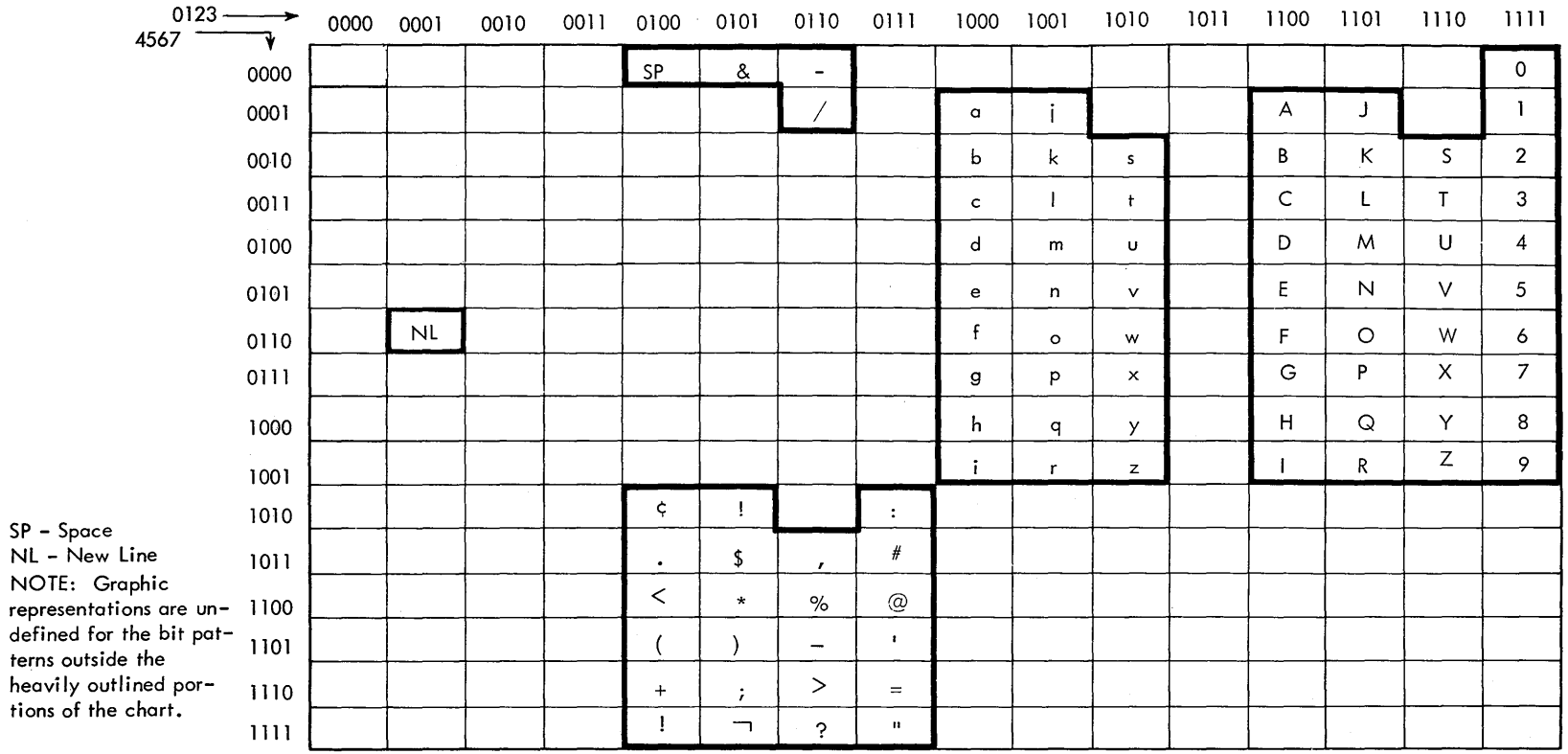


Figure C-16. EBCDIC Code Set

- 1052 Adapter operational diagrams are FEMDM Diagrams 6-28 through 6-33.

- All commands are described within the framework of initial selection, data transfer, ending sequence.

This section presents the detailed circuit descriptions of the 1052 Adapter (by command). FEMDM Diagrams 6-28 through 6-33 present the information in positive logic form, with the sequence of the circuits in those diagrams corresponding to the I/O interface signal sequence for the command being described. The seven valid commands, and the command byte bit configuration of each, are as listed below:

Command	Bit Configuration								
	P	0	1	2	3	4	5	6	7
Test I/O	1	0	0	0	0	0	0	0	0
Sense	0	0	0	0	0	0	1	0	0
Control No-Op	1	0	0	0	0	0	0	1	1
Control Alarm (No-Op)	0	0	0	0	0	1	0	1	1
Write-Auto Carrier Return	1	0	0	0	0	1	0	0	1
Write-Inhibit Carrier Return	0	0	0	0	0	0	0	0	1
Read	1	0	0	0	0	1	0	1	0

All other command byte bit configurations with correct parity are considered to be invalid commands.

WRITE

- Initial selection sequence

- Data transfer sequence

- Ending sequence

Initial Selection Sequence

The initial selection sequence for the write operation (write-ICR or write-ACR), begun by the channel, is illustrated in Diagram 6-29, FEMDM. The channel raises the 'select out', 'hold out', and 'address out' lines and places the address byte of the 1052 adapter on the 'bus out' lines. The adapter matches the address byte against the internally plugged address, and, if the two match, the 'address match' line is activated and ANDed with 'address out' to turn on both the 'initial select' trigger and the 'address in' trigger. The on-output of the 'address in' trigger turns on the 'operational in' trigger to raise the 'operational in' line to the channel.

The channel drops the 'address out' line when it receives the 'operational in' line from the adapter. When 'address out' drops, the adapter raises the 'address in' line to the channel and gates the output of the address bit generator to the 'bus-in' lines, thereby generating the address-in byte.

The channel receives the address byte and the 'address in' line and responds by placing a command byte on the 'bus out' lines and raising the 'command out' line. The 'command out' line from the interface (command-out cable out) ANDs with the on-output of the 'operational in' trigger to raise the 'command out' line with the adapter. A few nanoseconds after 'command out' is raised, 'the command out delay' line becomes active. The delay is the result of passing 'command out' through two inverters, each of which introduces a few nanoseconds delay into the signal.

'Command out' sets the operational-in interlock and activates the 'command gate'. 'Command gate' raises 'sense gate', and 'sense gate', in turn, raises 'write gate'. 'Write gate' then sets the 'write command' latch (the bit configuration of the command byte having activated valid command and write command bits) and the 'service request' latch.

'Command out delay' resets the address-in trigger, dropping address-in to the channel. When the channel recognizes the dropped 'address in' line, it drops 'command out'. The inactive 'command out' line now turns on the 'status in' trigger in the adapter. The 'status in' trigger raises 'status in' to the channel and gates the status byte to the 'bus in' lines.

The channel receives the 'status in' line and the status byte and replies with 'service out', indicating its acceptance of the status byte. The initial selection sequence is complete at this point.

Data Transfer Sequence

The 'service request' latch is set during the initial selection sequence (Diagram 6-29), but its on-output is not used until the end of the initial selection sequence. The 'service request' line in Diagram 6-29 activates the 'status conditions' line. 'Status conditions' activates the 'request in' line to the channel as soon as the 'operational in' trigger is

reset. The 'request in' line causes the channel to raise the 'hold out' and 'select out' lines as during the initial selection sequence. The channel, however, does not raise 'address out', nor does it place an address byte on the 'bus out' lines. This allows the 'address in' trigger to be set, but not the 'initial select' trigger. The adapter responds with 'address in' and its address byte as before, and channel, in turn, activates 'command out' to the adapter.

The channel does not place a command byte on the 'bus out' lines during the data transfer sequence; therefore, the 'write gate' (Diagram 6-29) is not activated in the adapter, and the 'service request' latch is not set by 'command out' during the data transfer sequence. The 'write command' latch is set during the initial selection and remains set through the data transfer sequence.

The 'busy condition' latch (Diagram 6-30) is set when the 'initial selection' trigger is reset by ANDing 'not initial select' trigger with 'write command'. The 'busy condition' latch is set at the end of the initial selection sequence. The on-output of the 'busy condition' latch is then ANDed with 'service request', 'operational in' trigger, 'not address in' trigger, and 'not command or service out' to set the 'service in' trigger when 'command out' falls during the data transfer sequence. The 'service in' trigger raises 'service in' to the channel, indicating the adapter is ready for a data byte.

The channel replies to 'service in' with 'service out' and places a data byte on the 'bus out' lines. The adapter ANDs 'service in' and 'service out' to activate 'service response' (Diagram 6-30) and start the read/write clock at SS 1. SS 1 gates the data byte from the 'bus out' lines to the data register. The clock runs through SS 7 to complete the operation by gating the output of the data register to and through the printer translator and/or function decoder.

SS 2 is ANDed with not 'printer busy', 'write', and 'not stop' to activate 'turn off service in' to reset the 'service in' trigger and drop the 'service in' line to the channel (Diagram 6-30). Channel then drops 'service out', and the data transfer sequence is complete. SS 6 is ANDed with 'not end of line', 'write', 'not busy or stop', and 'not service in and service out' to activate 'turn on service in' (Diagram 6-29). The active 'turn on service in' line sets the 'service request' latch, and 'service request' activates status conditions (Diagram 6-30) to begin another data transfer sequence by raising 'request in' to the channel. The data transfer sequences are continued until an ending sequence occurs in place of a data transfer sequence.

Ending Sequence

The Write-ICR and Write-ACR commands are distinguished by the ending sequences. The 'inhibit carrier return' latch is set during the initial selection for the Write-ICR command, allowing the adapter to present both the channel-end and the device-end status bits to the channel during the same ending sequence.

The Write-ACR command does not set the 'inhibit carrier return' latch during the initial selection. Therefore only the channel-end status bit is presented to the channel during the first ending sequence. The carrier is started moving to the left margin at about the same time the channel-end status is presented to the channel. A second ending sequence is required to present the device-end status bit to the channel when the carrier reaches the left margin and stops moving.

WRITE-ICR

The 'inhibit carrier return' latch is set during the initial selection sequence. 'Write gate' (activated by 'command out') ANDs with 'not bus out 4' to set the 'inhibit carrier return' latch during the time the command byte is on the 'bus out' lines (Diagram 6-31). The 'request in' line is activated to the channel at the end of the previous data transfer sequence (Diagram 6-30). The channel, however, has no more data to send to the adapter (channel byte count has gone to zero). The following signal sequence occurs: channel raises 'hold out' and 'select out'; adapter raises 'operational in' and 'address in' and places its address byte on the 'bus in' lines; channel raises 'command out'; adapter raises 'service in'. Thus far, the sequence is the same as for a data transfer sequence. Now, however, channel raises 'command out' a second time as a reply to 'service in', instead of 'service out'. A 'command out' reply to 'service in' indicates stop.

Diagram 6-31 illustrates the ending sequence for both Write-ICR and Write-ACR. 'Service in' and 'command out' are ANDed together to activate 'write/read turn on channel end'. 'Write/read turn on channel end' sets both the 'channel end' and the 'stop' latches. The on-output of the 'stop' latch sets the 'busy condition' latch and the 'store device end' latch. The on-output of the 'store device end' latch activates 'device end interrupt', which sets the 'device end' latch.

In Diagram 6-30, the 'service in' trigger is reset when the channel replies to 'service in' with 'command out'. Channel then drops 'command out' when 'service in' drops. In Diagram 6-31, the 'channel end' latch activates the 'status conditions' line, and 'status conditions' ANDs (in Diagram 6-29) with 'operational in' trigger, 'not address in' trigger, 'not service in' trigger, and 'not command or service out' to set the 'status in' trigger and raise the 'status in' line to the channel. The on-outputs of the 'status bit' latches are gated to the 'bus in' lines at the same time.

The channel replies with 'service out' to 'status in' from the adapter, indicating the acceptance of the status byte.

'Device end' ANDs with 'status in' (Diagram 6-30) to activate 'command reset'. The 'write command' latch and the 'inhibit carrier return' latch are reset by 'command reset'.

WRITE-ACR

The ending sequence for the Write-ACR operation begins the same as the ending sequence for Write-ICR. The major difference lies in the effect of not setting the 'inhibit carrier return' latch during the initial selection. When the ending sequence proceeds to where the channel replies to 'service in' with the second 'command out', indicating 'stop', the 'write/read turn on channel end' (Diagram 6-31) sets both the 'channel end' and 'stop' latches. The 'channel end' latch again activates the 'status conditions' line, and the 'status conditions' line again sets the 'status in' trigger (raising 'status in' to the channel) when 'command out' falls the second time.

Because the 'inhibit carrier return' latch is off, the 'carrier return' latch is set ('stop' ANDed with 'not printer busy' and 'not inhibit carrier return' (Diagram 6-30), and the read/write clock is started at cycle time. Cycle time then sets the 'printer busy' latch, picks the carrier return-line feed magnet in the printer, and picks the lowercase shift magnet in the printer if the printer was in upper case. The 'printer busy' latch now ANDs with 'stop' and 'carrier return' to set the 'inhibit carrier return' latch (Diagram 6-31).

Up to this point, the channel end status bit has been sent to the channel and the carrier has started returning to the left margin, but the device-end status bit has not been sent to the channel. The 'turn on device end' line becomes active (to set the 'store device end' latch Diagram 6-31) when the carrier has finished returning to the left and the 'busy' latch is reset. The on-output of the 'store device end' latch then activates 'device end interrupt', and 'device end interrupt' in turn activates the 'status conditions' line to raise 'request in' to the channel (Diagram 6-31).

The following sequence occurs: adapter raises 'request in' to the channel, channel raises 'hold out' and 'select out'; adapter raises 'operational in' and 'address in' and places its address byte on the 'bus in' lines; channel raises 'command out'; adapter drops 'address in'; channel drops 'command out'; adapter raises 'service in'; channel again raises 'command out'; adapter drops 'service in'; channel drops 'command out'. Up to this point, the sequence is the same as when the channel-end status bit was sent to the line. The difference is that this sequence was begun by the 'device end interrupt' line raising 'request in' to the channel.

The 'device end interrupt' line has been active during the above sequence, and 'device end interrupt' has kept the 'status conditions' line active. When channel drops 'command out' the second time, the 'status in' trigger (Diagram 6-29) is set by the 'status conditions' line. 'Status in' is raised to the channel, and the on-output of the 'device end' latch is gated to the 'bus in' lines as the 'device end' status bit. Channel rejects the status by replying to 'status in' with 'command out', and the status is stacked.

READ

- Initial selection sequence
- Data transfer sequence
- Ending sequence

Initial Selection

Initial selection for the Read command is almost identical with the initial selection for the Write command. The 'inhibit carrier return' latch is not set during the initial selection sequence because the read command byte contains the 4-bit. The 'read command' latch (Diagram 6-29) is set while 'command out' is up and the read command byte is on the 'bus out' lines. Note that the 'service request' trigger is not set at command-out time during the initial selection sequence. Recall that for the Write command the 'service request' latch was set by 'command out' and the write command byte. This led to the first data transfer sequence.

Data Transfer

The 1052 keyboard is unlocked when the 'read command' latch is set. Diagram 6-32 illustrates the circuits used for the data transfer sequence of the Read command. When a key is operated at the keyboard, the 'keyboard strobe' line becomes active and is ANDed with 'read' and 'not 1052 busy' to start the read/write clock at SS 5. SS 5 resets the data register, and SS 6 sets the output of the keyboard translator into the data register (Figure C-3). SS 7 ANDs with 'not shift change', 'keyboard strobe', and 'not stop' to activate 'turn on service in' to set the 'service request' latch (Diagram 6-32). 'Service request' in turn activates 'status conditions', and 'status conditions' raises the 'request in' line to the channel.

The channel replies to the 'request in' with 'select out' and 'hold out', and the data transfer sequence proceeds the same as the data transfer sequence for the Write command, with one exception. For the Write command, data is placed on the 'bus out' lines by the channel when the channel raises 'service out'. For the Read command, the adapter places data on the 'bus in' lines when the adapter raises

'service in'. The last character remains stored in the data register since the register is not cleared until a general reset occurs or until SS 5 time occurs during the following keyboard strobe.

With respect to Diagram 6-31, 'select out' ANDs with 'request in' to set the 'address in' trigger; the 'address in' trigger, in turn, sets the 'operational in' trigger, raising 'operational in' to the channel. Because 'address out' is not active when 'operational in' comes up, the adapter raises 'address in' and gates its own address to the 'bus in' lines. Channel replies to 'address in' with 'command out', and 'command out delay' drops the 'address in' line by resetting the 'address in' trigger. Channel drops 'command out' because the adapter drops 'address in'. The 'service in' trigger is set when 'command out' falls, raising the 'service in' line to the channel and gating the output of the data register to the channel.

Channel responds to 'service in' with 'service out', and the adapter ANDs 'service in' with 'service out' to activate 'service response' to start the read/write clock at cycle time. Cycle time causes the printer either to print the character in the data register or to perform the function called for by the character in the data register. The output of the keyboard has now been transferred to the channel and the printer, and the sequence is complete.

Ending Sequence

The ending sequence is similar to that for the Write-ACR command but can be started three different ways. First, the adapter can try to send one more data byte to the channel after the channel byte count has gone to 0. In this event, the sequence described under data transfer proceeds to the point where the character is stored in the data register and the adapter raises 'request in'. Channel makes a reply to 'service in' from the adapter. Instead of replying with 'service out' as during the data transfer sequence, channel replies with 'command out' for the second time, indicating stop. From this point on, the ending sequence is identical to the ending sequence for the Write-ACR command. Service response is not generated (Diagram 6-31) to start the read/write clock because 'service out' is not given as a reply to 'service in'. The last character stored in the data register is not printed because the clock is not started, nor is it accepted by the channel.

The second means of obtaining the end sequence for the Read command is through the use of the 'enter' signal. When the operator has finished keying in data, if the byte count in the channel has not previously gone to zero, he may end the read operation by depressing the ENTER pushbutton. This causes the 'enter' line in Diagram 6-31 to activate the 'write/read turn on channel end' line. From this point on, this ending sequence proceeds exactly the same as the ending sequence described in the preceding paragraph.

The third means of obtaining the ending sequence for the Read command is through the use of the 'cancel' signal. If the operator has made a keying error, he may cause the channel to ignore the block of data in which the error was made. The operator depresses the CANCEL pushbutton. This activates the 'cancel' line in Diagram 6-31 and causes the ending sequence to proceed as described in the first paragraph of this section.

Status Byte Composition

The status byte transferred to the channel may be composed of any six bits. Diagram 6-33 illustrates the composition of the status byte. Bit position 0 indicates the REQUEST pushbutton on the 1052 has been operated. Bit position 3 indicates that the channel has attempted to execute an initial selection sequence with the adapter while the adapter was busy executing another command. Bit position 4 indicates the adapter has finished executing a command and the channel has not yet been made aware of this condition. Bit position 5 indicates the 1052 has finished executing a command and the channel has not yet been made aware of this condition. Bit position 6 indicates an error condition has occurred. Bit position 7 indicates the CANCEL key has been operated (Read command only). Bit positions 1 and 2 are not used in the status byte.

The individual error conditions indicated by bit position 6 are not defined in the status byte. The channel must issue a Sense command to determine the particular cause of the error indication.

SENSE COMMAND

- Initial selection sequence
- Sense byte transfer sequence
- Ending sequence (a continuation of the sense byte transfer)

The Sense command is normally issued following an error indication in the status byte of an ending sequence for either a Read or Write command. If the status byte contains the unit check (6) status bit, the channel usually issues a Sense command to determine the cause of the error indication.

Initial Selection Sequence

The initial selection sequence for the sense command is almost identical with the initial selection sequence for the

Write command. When the sequence proceeds to the point where the channel raises 'command out' and places the Sense command on the 'bus out' lines, the adapter sets the operational-in interlock, the 'sense command' latch, and the 'service request' latch (Diagram 6-29). The 'operational in' trigger is reset at the end of the initial selection sequence, and the adapter disconnects from the channel.

Sense Byte Transfer Sequence

The adapter raises the 'request in' line to the channel when the 'operational in' trigger is reset because the 'service request' latch is on. The channel and adapter then proceed through a signal sequence identical with a read data transfer sequence to the point where the adapter raises 'service in' to the channel. Diagram 6-33 shows that when the adapter raises 'service in' with the 'sense command' latch on, the 'gate sense in' line is activated, gating the on-output of the 'command reject', 'bus out check', and 'equipment channel check' latches to the 'bus in' lines. The off-output (on-output inverted) of the 'ready' latch is also gated to the 'bus in' lines at this time. The configuration of the sense byte then indicates just what condition caused the 6-bit to be set in the status byte.

Ending Sequence

For the Read and Write commands, the adapter disconnects from the multiplexer channel after a byte is transferred to or from the channel. This disconnect does not occur between the sense byte and the final status byte for the Sense command. The 'operational in' trigger is reset when the operational-in interlock turns off (not operational-in interlock) (Diagram 6-29) but the 'command' latch is not reset by 'service in' ANDed with 'service out' while the 'sense command' latch is on (Diagram 6-29).

In Diagram 6-31 'sense command' is ANDed with 'service in' and 'service out' to set both the 'channel end' and 'device end' latches. The 'channel end' latch activates the 'status conditions' line, and (in Diagram 6-29) the 'status conditions' line sets the 'status in' trigger when the 'service in' trigger is reset.

Therefore, as soon as the channel accepts the sense byte by replying to 'service in' with 'service out', the adapter sets the 'channel end' and 'device end' latches (Diagram 6-31), resets the 'service in' trigger (Diagram 6-30), and sets the 'status in' trigger (Diagram 6-29) to present the channel end and device end status bits to the channel. When the channel replies to the 'service in' with 'service out', the operation is complete.

CONTROL COMMANDS

There are two Control commands: Control Alarm and Control No-Op. Both cause only an initial selection sequence to take place. The channel end and device end status bits are included in the initial selection status byte. Initial selection for these two commands proceeds the same as the initial selection for the write commands to the point where the channel raises 'command out' and places the command byte on the 'bus out' lines.

When the adapter raises the sense gate (Diagram 6-29), the control command byte bit configuration activates the 'control' line (Diagram 6-31) to set both the 'channel end' and the 'device end' latches. These two latches being on when the 'status in' trigger (Diagram 6-29) is set allows the status byte to contain both the channel end and device end status bits.

The only difference between the two commands is the 4-bit. The Control Alarm command contains the 4-bit, while the Control No-Op command does not. Control No-Op is a programming aid producing no useful function within the adapter or 1052. Control Alarm activates the general selective initial reset and resets the sense byte. Diagram 6-31 illustrates the 'control alarm' line that initiates the resets.

TEST I/O

The Test I/O command is issued to interrogate the status of the adapter. An initial selection sequence will complete the operation. The channel raises 'service out' in reply to 'status in' to end the operation.

The 'test I/O' latch (Diagram 6-29) is set during the time that 'command out' is up. 'Not initial select' trigger resets the 'test I/O' latch as soon as the initial selection sequence is completed. If the Test I/O command is issued while the adapter is executing a Read, Write, or Sense command the channel control examines the unit control word and recognizes that a previous operation is in progress. Initial selection does not take place; the channel itself sets the busy bit.

Test I/O clears any of the following pending interrupt conditions: attention status device end status, channel end status, and status stacked. The busy bit is not included in the status byte when any of these four conditions are cleared.

HALT I/O

Halt I/O is not a command in the same sense as Read, Write, Sense, Test I/O, and Control. These five commands

result in at least an initial selection sequence, with the channel placing a command byte on the 'bus out' lines while raising the 'command out' line. Halt I/O is, instead, a condition of the interface lines where 'address out' and 'operational in' are up while 'select out' is down.

This condition is generated by the channel to stop a read or write operation currently in progress. The halt I/O function (Diagram 6-31) is activated by 'operational in' trigger ANDed with 'address out' and 'not select out'. These three conditions result in activating the 'turn on stop, busy, channel end' line. This line sets the 'stop', 'busy condition', and 'channel end' latches. The halt I/O function activates the 'general', 'selective', or 'I/O disconnect reset' line (Diagram 6-31), resetting the 'operational in' and 'initial select' triggers (Diagram 6-29), the 'service request' latch (Diagram 6-29), and the 'service in' trigger (Diagram 6-30).

The 'channel end' latch (Diagram 6-31) raises 'request in' to present channel end status to the channel. This channel end interrupt is cleared by the channel accepting the channel end status byte by replying to 'status in' with 'service out'.

GENERAL OR SELECTIVE RESET

The general or selective reset lines are illustrated in Diagram 6-31. Selective reset affects only the control unit whose 'operational in' trigger is on. This reset then affects only the one control unit connected to the channel when 'operational out' is dropped. General reset affects all control units on the channel, since the 'operational in' trigger is not one of the limiting conditions.

This appendix discusses: (1) the hexadecimal (hex) number system, (2) the eight-bit zoned character codes, (3) the instruction formats and operand designations, and (4) the various data formats.

HEXADECIMAL NUMBER SYSTEM

- System uses 16 symbols: 0–9, A–F.
- Base of system is 16.
- System is shorthand notation for binary numbers.
- Four binary bits are represented by one hex symbol.
- Byte is represented by two hex symbols.

Binary numbers have approximately 3.3 times as many terms as their decimal counterparts. This increased length presents a problem when talking or writing about binary numbers. A long string of 1's and 0's cannot be effectively spoken or read. A shorthand system is necessary, one that has a simple relationship to the binary system and that is compatible with the basic eight-bit byte used in the CPU. The hexadecimal (hex) number system meets these requirements.

The hex system has 16 symbols: 0–9, A–F. Counting is performed as in the decimal and binary systems. When the last unique symbol (F) is reached, a 1 is placed in the next position to the left and counting resumes with a 0 in the original position, as follows:

0	10	20	A0
1	11	21	A1
2	12	22	A2
3	13	23	↓
4	14	↓	
5	15		
6	16		and so on
7	17		
8	18		
9	19		
A	1A	9A	
B	1B	9B	
C	1C	9C	
D	1D	9D	
E	1E	9E	
F	1F	9F	

One hex symbol can represent four binary bits. Thus the 8-bit binary byte, in turn, can be represented by two hex symbols. The relationship between the hex, binary, and decimal systems is as follows:

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

The important relationship to remember is that four binary positions are equivalent to one hex position.

Hex numbers are represented in the same manner as decimal and binary numbers, except that the base is 16. The terms of the number represent the coefficients of the ascending powers of 16. For example, consider the hex number 257 (decimal equivalent equals 599):

$$257 = (2 \times 16^2) + (5 \times 16^1) + (7 \times 16^0)$$

$$= (2 \times 256) + (5 \times 16) + (7 \times 1)$$

$$= 512 + 80 + 7$$

$$= 599_{10}$$

EIGHT-BIT ZONED CHARACTER CODES

All I/O devices requiring a zoned character code use either the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) or the USA Standard Code for Information Interchange extended to eight bits (USASCII-8). The EBCDIC and USASCII-8 codes for the hex characters 0–F are listed below. For charts showing the complete EBCDIC and USASCII-8 codes with associated graphic characters, refer to 9020D/E Principles of Operation. The codes do not have a symbol defined for all 256 eight-bit codes. To represent codes that do not have a defined symbol, two hex

terms (representing four bits each) may be used instead of the eight-bit code.

Hex Code	Printed Graphic	EBCDIC Code	USASCII-8 Code
0000	0	1111 0000	0101 0000
0001	1	1111 0001	0101 0001
0010	2	1111 0010	0101 0010
0011	3	1111 0011	0101 0011
0100	4	1111 0100	0101 0100
0101	5	1111 0101	0101 0101
0110	6	1111 0110	0101 0110
0111	7	1111 0111	0101 0111
1000	8	1111 1000	0101 1000
1001	9	1111 1001	0101 1001
1010	A	1100 0001	1010 0001
1011	B	1100 0010	1010 0010
1100	C	1100 0011	1010 0011
1101	D	1100 0100	1010 0100
1110	E	1100 0101	1010 0101
1111	F	1100 0110	1010 0110

INSTRUCTION CODING

The 7201-02 CE uses the Universal instruction set, which enables it to execute fixed-point, floating-point, decimal, logical, branching, status switching, and I/O instructions. In addition, special 9020D/E multiprocessing instructions and 9020E display instructions have been designed into the CE.

The total instruction set uses five instruction formats: RR, RX, RS, SI, and SS. Operands are designated as first, second, or third operands. For addressing purposes, the operands are grouped into three classes:

1. Effectively addressed operands in main storage.
2. Immediate operands in the instruction format.
3. Operands in local storage (LS).

Instruction Formats

- Five instruction formats are available: RR, RX, RS, SI, and SS.
- Halfword is basic building block for instruction.
- Instructions are made up of 1, 2, or 3 halfwords.
- First halfword contains 8-bit op code and, depending on format:
  4-bit LS address for operand, or operand address component.
  4-bit mask.
  8-bit immediate operand.
  4-bit or 8-bit length fields.
- Second and third halfwords contain:
  4-bit LS address for operand address component.
  12-bit displacement.

Five instruction formats are available, denoted by the format codes RR, RX, RS, SI, and SS. The format codes express, in general terms, the operation to be performed. RR denotes a register-to-register operation; RX, a register-to-indexed-storage operation; RS, a register-to-storage operation; SI, a storage and immediate-operand operation; SS, a storage-to-storage operation. The overall instruction set for the CE may be divided into nine classes; the breakdown by format is as follows:

Instruction Class	Format
Fixed-Point	RR, RX, RS
Floating-Point	RR, RX
Decimal	SS
Logical	RR, RX, RS, SI, SS
Branching	RR, RX, RS
Status Switching	RR, SI
I/O	SI
Multiple Computing Element	RR, SI, SS
Display	RR, RX

The basic unit of length for instructions is the halfword, consisting of two bytes. The length of an instruction format can be 1, 2, or 3 halfwords. It is related to the number of initial main storage references necessary for the operation. An instruction making no reference to main storage (RR) consists of one halfword; an instruction making one reference (RX, RS, or SI) consists of two halfwords; an instruction making two references (SS) consists of three halfwords. All instructions must be located in main storage on an integral boundary for halfwords. The five formats are shown in Figure D-1.

For purposes of describing the execution of instructions, operands are designated as first, second, or third operands, referring to the manner in which the operands participate in the operation. The operand to which a field in an instruction format pertains is denoted by the number following the letter designation of the field; for example, the R1 field is the address of an LS register containing the first operand; R2, the second operand.

As shown in Figure D-1, the first halfword of each format consists of two parts. The first byte contains the operation code (op code), which identifies the operation to be performed. Bits 0 and 1 specify the format, bits 2 and 3 specify the class of instruction, and bits 4 through 7 identify the instruction within the class. The second byte of

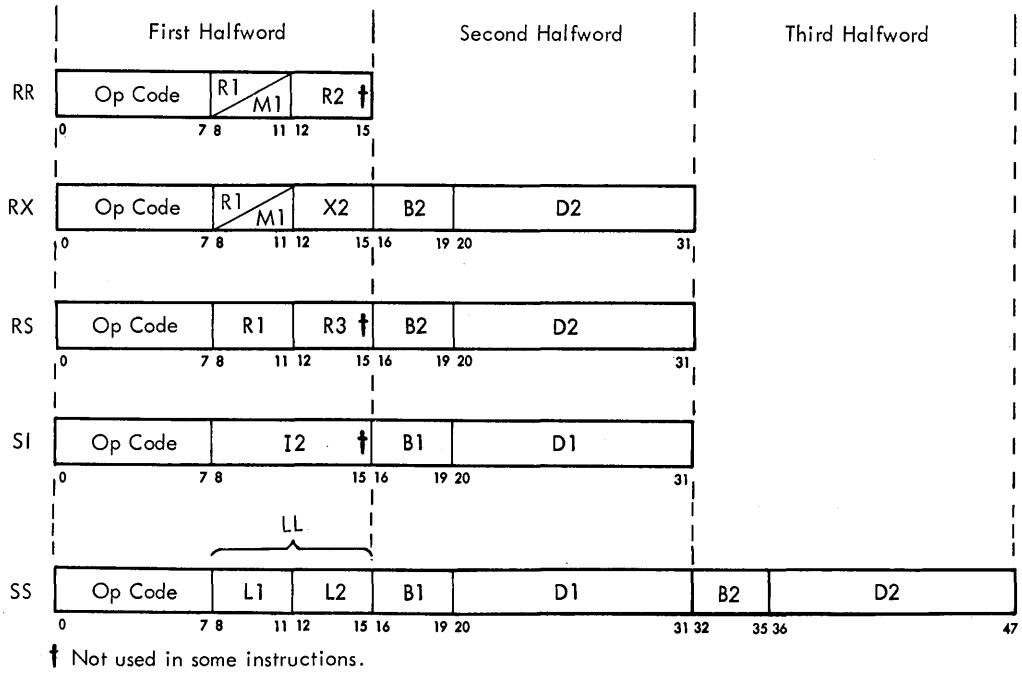


Figure D-1. Instruction Formats

the first halfword is used as either two four-bit fields or a single eight-bit field. The fields and the information contained within the fields are as follows:

1. R1, R2, and R3: four-bit address of an LS register containing the first, second, and third operands, respectively.
2. M1: four-bit mask used in some branching instructions.
3. X2: four-bit address of an LS register containing the index value used in generating the effective second operand address.
4. I2: eight-bit byte of immediate data (second operand).
5. L1 and L2: four-bit length (up to 16 bytes) of first and second decimal VFL operands, respectively.
6. LL: eight-bit length field (up to 256 bytes) for logical VFL operands.

The second and third halfwords always contain the same information: a four-bit address of an LS register containing the base value to be added to the following 12-bit displacement field.

Operand Addressing

For addressing purposes, operands are grouped into three classes: (1) effectively addressed operands in main storage, (2) immediate operands in the instruction format, and (3) operands in LS.

Effectively Addressed Operands

- Operands are either fixed-length or VFL.
- Fixed-length operands are located on integral boundary.
- Length of VFL operand is specified by L or LL field.
- L and LL fields denote number of bytes to right of addressed byte.
- Effective operand address is sum of 24-bit base address, 12-bit displacement, and 24-bit index value.
- Base address and index value are located in LS.
- Displacement is located in instruction format.

An effectively addressed operand is selected from a main storage location not related to the location of the instruction referring to it. It is specified by means of a main storage address. When the operand consists of more than one byte, the address gives the location of the first byte; subsequent bytes are located in the next-higher addressed byte locations. Both the first and second operands of an instruction can be effectively addressed.

Effectively addressed operands can be of either fixed length or variable field length. The length of VFL operands, in terms of the number of bytes to the right of the addressed byte, is specified by the L or LL field of the instruction. The LL field can be eight bits long and thus can specify a maximum operand field length of 256 bytes.

In the instruction format, an effectively addressed operand is specified by a base address, a displacement, and, in some cases, an index value. The base address and the index value are contained in LS general-purpose registers addressed by the B and X fields, respectively, of the instruction. The registers contain 32 bits, the low-order 24 of which constitute an unsigned address component (base address or index value). The high-order eight bits of the

register are ignored. The 24-bit base address is included in every address computation. The 24-bit index value is included in the address computation as specified by the RX instruction format.

The displacement value is a 12-bit number contained in the D-field of the instruction. It is included in every address computation. The displacement provides for relative addressing up to 4095 bytes beyond the base address.

In computing the effective operand address, the base address and the index value are treated as 24-bit positive binary integers having no sign position. The displacement is similarly treated as a 12-bit positive binary integer. The three numbers are added. Because every operand address includes a base address, the sum is always 24 bits long. Any overflow above the 24 low-order bits of the sum is ignored, causing a lower address to be generated. If this lower address is above the maximum available storage, an addressing program interruption occurs. If the lower address is available the CE accesses that location.

An instruction may contain zeros in the B, X, or D field. In the case of the B and X fields, a zero does not denote the address of LS general-purpose register 0, but indicates that base and index values of zero are to be used in generating the effective operand address. Similarly, a D field of zero indicates a displacement of zero.

Fixed-length fields, halfwords, words, and doublewords, must be located in main storage on an integral boundary for that length field. A boundary is called "integral" for a field when its storage address is a multiple of the length of the field in bytes (Figure D-2). For example, words (four bytes) must be located in main storage so that their address is a multiple of the number 4. A halfword (two bytes) must have an address that is a multiple of 2, and doublewords (eight bytes) must have an address that is a multiple of 8.

Main storage addresses are expressed in binary form. Therefore, integral boundaries for halfwords, words, and doublewords can be specified only by binary addresses in which 1, 2, or 3 of the low-order bits, respectively, are zero (Figure D-2). Thus, integral boundaries for words are binary addresses in which the two low-order bit positions are zero; for example, 00000, 00100, 01000, and 01100.

VFL fields are not limited to integral boundaries, but may start on any byte location.

Immediate Operands

Immediate operands are contained in SI instructions for logical operations. They are one byte (bits 8–15) long, serve as the second operand, and are designated I2.

Operands in Local Storage

- LS registers are addressed by four-bit R-field in instruction format.
- LS GPR's are addressed 0–15.
- LS FPR's are addressed 0, 2, 4, and 6.
- For fixed-point doubleword operands, the register address implies use of a pair of adjacent registers.

Fixed-point and floating-point operands may be located in the 16 general-purpose registers (GPR's) and the 4 floating-point registers (FPR's), respectively, of LS. The registers are addressed by a four-bit field in the instruction, designated the R-field. The GPR's are designated by addresses 0–15, whereas the FPR's are identified by addresses 0, 2, 4, and 6. (When an FPR is designated by any other address, a specification program interruption occurs.) The op code of the instruction implies whether the GPR's or the FPR's are to be used.

The GPR's are 32 bits (one word) in length. Fixed-point operands normally have an implied length of one word. In some operations, one register address implies the use of a pair of adjacent GPR's, thus providing a doubleword. For these instructions, the addressed register (say R1) contains the high-order operand bits and must have an even address, and the implied register (R1 + 1) contains the low-order operand bits and has the next higher address.

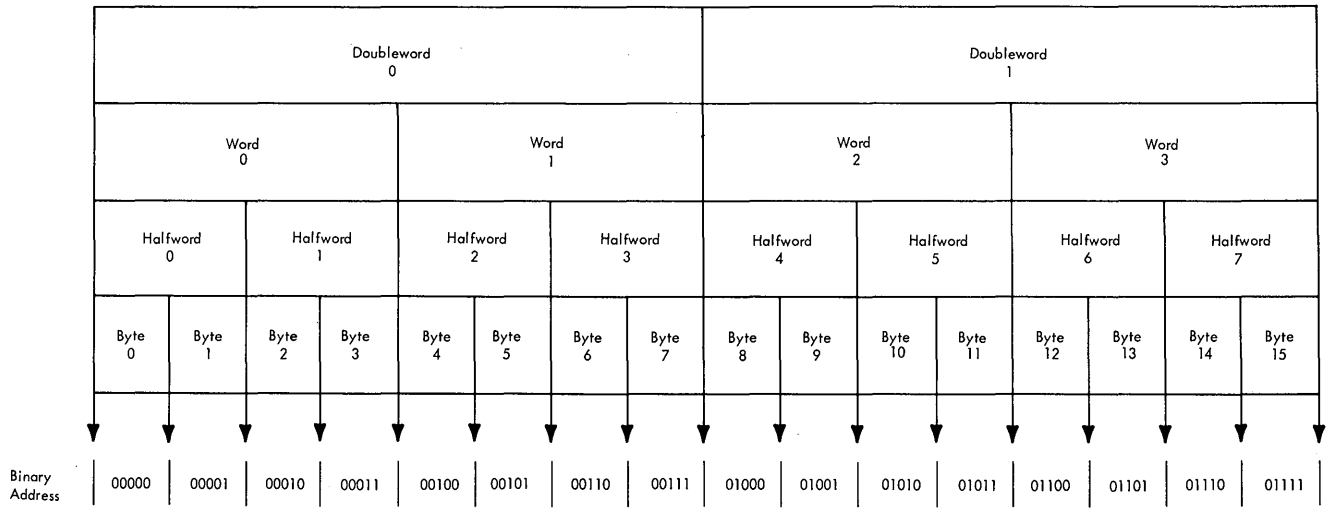


Figure D-2. Main Storage Integral Boundaries

The FPR's are 64 bits or a doubleword in length, and can contain either a short (one word) or a long (doubleword) floating-point operand. A short operand occupies the high-order bits of an FPR; the low-order bits are ignored.

DATA FORMATS

Data can be numeric, alphabetic, or logical, and fixed or variable in length. Numeric data is distinguished as fixed-point, floating-point, or decimal. The data may be divided into four classifications:

1. Fixed-point numbers, having a binary radix and a fixed length, usually a word or a halfword.
2. Floating-point numbers, represented by a 7-bit characteristic and a signed hex fraction, occupying either a word or a doubleword.
3. Decimal numbers, represented by four-bit binary-coded-decimal (BCD) digits, usually packed two digits to a byte, and variable in length.
4. Logical information, represented by eight-bit zoned character codes, and fixed or variable in length.

Fixed-Point Data

Fixed-point instructions are available for loading, adding, subtracting, comparing, multiplying, and dividing. A pair of conversion instructions, Convert to Binary and Convert to Decimal, provides transition between decimal and binary radix without the use of tables.

The basic fixed-point operand is the 32-bit binary word. To improve performance and storage utilization, 16-bit halfword operands may be specified in most operations. In both lengths, bit position 0 holds the sign of the number, with the remaining bit positions designating the magnitude of the number. To preserve precision, some products and all dividends are 64 bits long.

Because a 24-bit address can be accommodated in the 32-bit word, fixed-point instructions can be used both for integer operand arithmetic and for address computation. This combined usage provides economy and permits the entire fixed-point instruction set to be used for address computation. Thus, multiplication, shifting, and logical manipulation of address components are possible.

Number Representation

- Positive numbers are represented in true binary form with sign of 0.
- Negative numbers are represented in 2's complement notation with sign of 1.

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary form with a sign bit of 0. Negative numbers are represented in 2's complement notation with a sign bit of 1. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit; i.e., all 0's for positive numbers, all 1's for negative numbers. Therefore, when an operand must be extended with high-order bits, each nonsignificant bit is made equal to the sign bit.

Negative fixed-point numbers are formed in 2's complement notation by complementing the true binary representation of the number and adding 1. For example, to convert the decimal number +26 to 2's complement form (–26), proceed as follows:

	S 1◄—Integer—►31
Decimal +26 to true binary form:	0 0000000 00011010
Complement the binary number:	1 1111111 11100101
Add 1:	1
Result is -26 (2's complement form):	1 1111111 11100110

The result is equivalent to subtracting the number 00000000 00011010 from 1 00000000 00000000.

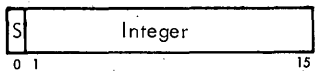
The largest positive number consists of a sign bit of 0 with all 1's in the integer field, whereas the largest negative number consists of a sign bit of 1 with all 0's in the integer field:

	S 1◄—Integer—►31	Decimal Number
Largest positive number:	0 1111111 1111111	= +2,147,483,647
Smallest positive number:	0 0000000 0000000	= 0
Smallest negative number:	1 1111111 1111111	= -1
Largest negative number:	1 0000000 0000000	= -2,147,483,648

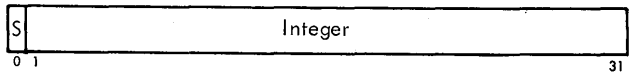
Formats

Fixed-point numbers occur in 16-bit halfword, 32-bit word, or 64-bit doubleword operands. Bit 0 is the sign bit, and the remaining bits make up the integer:

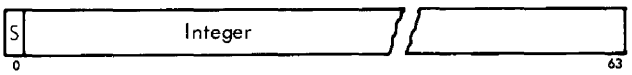
Halfword operand



Word operand



Doubleword operand



In LS, fixed-point operands are a word long. In some operations, such as multiply, divide, and shift, the operand may be a doubleword. The doubleword operands are located in a pair of adjacent 32-bit GPR's and are addressed by an even address that refers to the leftmost (lower-addressed) register of the pair. In this case, the sign-bit position of the rightmost register may contain an operand bit instead of a sign bit. The sign-bit position of the leftmost register must always contain a sign bit.

In main storage, fixed-point data may be a halfword, a word, or a doubleword. This data must be located on integral storage boundaries for these units of information. When a halfword operand is fetched from main storage, it is extended to a full word. The original signed integer occupies bits 16 through 31, and is operated on as a full word. When the operand is extended to a full word, bits 0 through 15 assume the state of the original sign bit, now in bit 16.

Floating-Point Data

- Scientific and engineering calculations require that very small and very large numbers be represented.
- Scientific notation uses powers of 10 to simplify calculations with high and low magnitude numbers.
- Floating-point instructions operate upon data that uses powers of 16 to represent numerical quantities.
- Quantity expressed by floating-point number is product of signed hex fraction and 16 raised to power designated by exponent.

When performing calculations for scientific and engineering work, very small or very large numbers must sometimes be represented. Consider the problem of a nuclear physicist who wants to write an equation that contains the value for the mass of a subatomic particle. If he wrote the number as a decimal fraction, he would have to put down a decimal point followed by more than 20 zeros and a few numerals. At the opposite extreme, an astronomer may be calculating distances between objects that may be millions of millions of miles apart.

To overcome the pointless effort of having to write many zeros when working with such numbers, a mathematical method using powers of 10 is used. This method is called scientific notation. For example, in scientific notation, the mass of an electron can be written as 9.107×10^{-28} grams; and the astronomical distance of one light year can be written as 5.878×10^{12} miles.

Consider the parts of a number represented in scientific notation. The example of the mass of an electron, for instance, can be divided into three distinct parts: (1) a signed mixed number (+9.107) multiplied by, (2) 10 raised to the power designated by, (3) a signed exponent (−28). By changing the position of the decimal point and adjusting the exponent to compensate for the change, the number can also be written as a fraction times a power of 10 ($+.9107 \times 10^{-27}$), or an integer times a power of 10 ($+9107 \times 10^{-31}$).

For scientific and engineering applications where quantities may be of the magnitudes mentioned above, the CE has instructions for handling them. These instructions, called floating-point instructions, manipulate data in a manner similar to scientific notation. However, because the CE is primarily a binary machine in which numbers can be easily worked upon in hex (four-bit) units, a quantity is represented as a hex number times a power of 16 rather than as a decimal number times a power of 10. Except for this difference in base, floating-point notation is similar to scientific notation; the same rules of algebra apply to powers of 16 as to powers of 10.

Number Representation

- Fraction represents number expressed in hex digits.
- Characteristic specifies exponent to which 16 is raised.
- Characteristic is expressed in excess 64 notation; range is −64 to +63.
- Radix point is to left of high-order hex digit.
- True zero result yields positive sign.

A floating-point number contains the same components as a number written in scientific notation. However, due to the nature of the computer, the format is different and certain rules are imposed upon the way a floating-point number may be represented. The number to be multiplied by a power of 16 is a hex fraction with a fixed length, and the 16 is understood rather than shown. Therefore, as represented in the CE, a floating-point number consists of a sign, which is the sign of the fraction, a signed exponent, called a characteristic, and a hex fraction. The quantity expressed by this number is the product of the fraction and 16 raised to the power designated by the characteristic (exponent).

The fraction of a floating-point number is expressed in hex digits. The radix point (representing the base 16) of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for a floating-point number, the fraction is considered to be multiplied by a power of 16 (fraction x 16^n power). The characteristic (bits 1–7) indicates the power (exponent).

Bit 0 designates the sign of the fraction; it is a 0 if the fraction is a positive number and a 1 if the fraction is negative. Both positive and negative quantities are in true form, with the difference indicated by the sign.

The exponent may also be either a positive or negative number. For example, $-.A8 \times 16^{-2}$ is an example of a floating-point number with a negative fraction and a negative exponent. Therefore, to represent both positive and negative exponents, excess 64 notation is used. Excess 64 notation simply means that +64 (+40 hex) is added to the true exponent and the value obtained is used as the characteristic. Therefore, the characteristic varies around a base of 64; i.e., an exponent of 0 is represented by a characteristic of 64, a positive exponent is represented by a characteristic greater than 64, and a negative exponent is represented by a characteristic less than 64. In the example just given, for instance, −2 is the exponent. Adding +64 to −2 yields +62, which is the value of the characteristic in excess 64 notation.

Performing the same calculation in binary gives the following results:

```
Bits     0 1 2 3 4 5 6 7

         S 1 1 1 1 1 1 0    2's complement of 2 (-2)
         S 1 0 0 0 0 0 0    +64 (+40 hex)
         S 0 1 1 1 1 1 0  = +62 (+3E hex)
           ↓
         Carry
```

If the exponent were +2 (positive exponent), adding +64 yields +66, the characteristic value placed into bits 1−7. In binary, the addition is as follows:

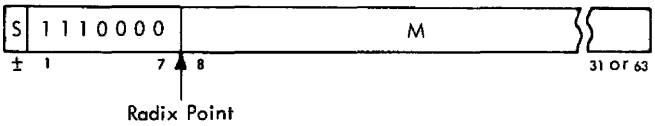
```
Bits     0 1 2 3 4 5 6 7

         S 0 0 0 0 0 1 0    +2 exponent
         S 1 0 0 0 0 0 0    +64 (40 hex)
         S 1 0 0 0 0 1 0  = +66 (42 hex)
```

Note in these examples that a negative exponent in excess 64 notation caused bit 1 (high-order bit of the characteristic) to be a 0, and a positive exponent caused bit 1 to be a 1. This rule holds true for the range of positive and negative exponents, as can be seen in Table D-1. The table also shows that because only seven binary bits (1−7) are available to represent the characteristic in floating-point format, the most negative exponent that can be expressed is −64 and is represented by an all-zero characteristic. The most positive exponent is +63, and is represented by all 1's (7F hex). Midpoint between these two extremes is a 0 exponent, which is represented by a +64 (+40 hex) characteristic.

Table D-1. Characteristic Notation

Excess 64 Notation			
Binary	Decimal	Hex	Exponent
0000000	0	0	-64
0000001	1	1	-63
↓	↓	↓	↓
0111111	63	3F	-1
1000000	64	40	0
1000001	65	41	+1
↓	↓	↓	↓
1111110	126	7E	62
1111111	127	7F	63

For another example of converting an exponent to an excess 64 characteristic, assume the value of $\pm M \times 16^{48}$ must be stated in excess 64 notation. The characteristic of the fraction then becomes 48 + 64 = 112 (0110000 + 1000000 = 1110000). The floating-point number thus takes the following form:



To illustrate how numbers are represented in floating-point format, assume that the decimal number 149.25 is to be converted to a floating-point short operand. This conversion is accomplished as follows:

1. The number is separated into a decimal integer and a decimal fraction:

   149.25 = 149 plus 0.25

2. The decimal integer is converted to its hex representation:

   $149._{10} = 95_{16}$

3. The decimal fraction is converted to its hex representation:

   $0.25_{10} = 0.4_{16}$

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent):

$$95.4_{16} = (0.954 \times 16^2)_{16}$$

5. The characteristic is developed from the exponent and converted to binary:

Excess 64 + exponent = characteristic
64 + 2 = 66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally:

$$.954_{16} = 1001\ 0101\ 0100$$

7. The characteristic and the fraction are placed in the short precision format; the sign position contains the sign of the fraction:

S	Characteristic	Fraction
0	1000010	1001 0101 0100 0000 0000 0000

Other examples follow:

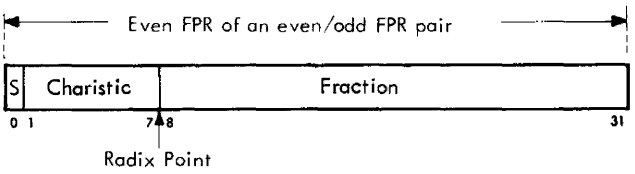
Number	Powers of 16	S	Charistic	Fraction
1.0	$= +1/16 \times 16^1$	= 0	1000001	0001 0000 ≈ 0000
0.5	$= +8/16 \times 16^0$	= 0	1000000	1000 0000 ≈ 0000
1/64	$= +4/16 \times 16^{-1}$	= 0	011111	0100 0000 ≈ 0000
0.0	$= +0 \times 16^{-64}$	= 0	0000000	0000 0000 ≈ 0000
-15.0	$= -15/16 \times 16^1$	= 1	1000001	1111 0000 ≈ 0000
2×10^{-78}	$= +1/16 \times 16^{-64}$	= 0	0000000	0001 0000 ≈ 0000
7×10^{75}	$= (1-16^{-6}) \times 16^{63}$	= 0	1111111	1111 1111 ≈ 1111

Formats

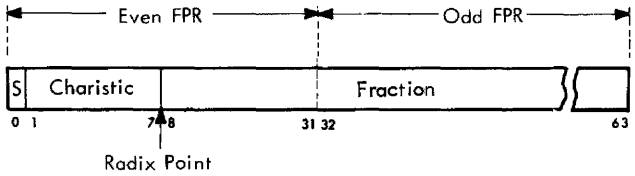
- Data format consists of 1-bit sign, 7-bit characteristic, and 24- or 56-bit fraction.
- Results are 32 bits (short operand) or 64 bits (long operand) long.
- Multiply product is always 64 bits.
- Guard digit is retained.

Floating-point data is represented in the CPU in one of two fixed-length formats, depending upon whether a full-word short operand or a doubleword long operand is desired. Both formats may be used in main storage and in the eight LS FPR's used exclusively by floating-point instructions. The data formats for short and long operands are:

Short Operand



Long Operand



For both formats, the first bit position is the sign bit and the subsequent seven bit positions constitute the characteristic. The following 24 or 56 bits represent the fraction; short operand fractions are 24 bits or 6 hex digits; long operand fractions are 56 bits or 14 hex digits.

When short operands are specified, the results are usually 32-bit floating-point words; the odd FPR of the even/odd pair of FPR's does not participate in the operation and remains unchanged. However, in multiply instructions, the product occupies two FPR's (64 bits).

When long operands are specified, all operands and results are 64-bit floating-point doublewords.

Although final results have 6 or 14 hex fraction digits, intermediate results in addition, subtraction, and compare operations may extend to 7 or 15 fraction digits. This extra digit, called the guard digit, occurs when one of the fractions is shifted right (as part of the characteristic equalization process that occurs during execution of add-type floating-point instructions; see Chapter 3, Section 3, Add, Subtract, and Compare). The guard digit increases the accuracy of the final result if normalization occurs. In normalization, the fraction is shifted left until a significant digit appears in the high-order digit position of the fraction; thus the guard digit becomes part of the 6 or 14 hex digits of the final result. This saving of the guard digit becomes especially significant where the high-order 6 or 14 digits of the intermediate result are all zeros.

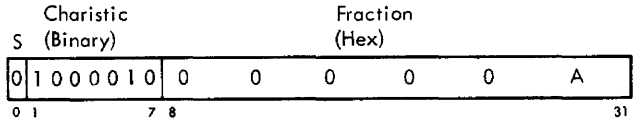
Normalization

- Normalized fraction has nonzero, high-order hex digit; unnormalized fraction has one or more leading hex zeros.
- Characteristic is adjusted on normalization cycles.
- Postnormalization is normalization of final result.
- Prenormalization is normalization before result computation.

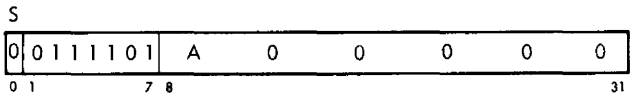
- Results are shifted right if fraction overflow occurs.

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hex fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. Normalization consists of shifting the fraction left until the high-order hex digit is nonzero and reducing the characteristic by the number of hex digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

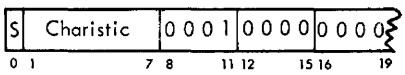
An example of an unnormalized floating-point number in numerical terms is $.00000A_{16} \times 16^{2}$. To convert this number to its normalized form, the number must be shifted five hex digits to the left and, because five shifts are necessary, five is subtracted from the exponent. The result is $.A00000_{16} \times 16^{-3}$. In the CPU, the original number would have the following format:



After normalization, the format would be:



Because normalization applies to hex digits, the three high-order bits of a normalized number may be zero. For example, if the high-order digit of a fraction is a hex 1, normalization will not occur although normalization is specified and bits 8–10 = 000:



Floating-point operations are performed with or without normalization. Addition and subtraction may be specified either way depending upon the instruction op code. The multiply, divide, and halve instructions always specify normalization. The load, compare, and store instructions specify unnormalized results. Normalization usually occurs when the intermediate arithmetic result is changed to the final result. This function is called postnormalization. In multiplication and division, the operands are normalized before the arithmetic process. This function is called prenormalization.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. In both normalized and unnormalized operations, the initial operands need not be in normalized form.

Decimal Data

- Operands and results are located in main storage.
- VFL is 1–16 bytes.
- Four-bit BCD digits are packed two to a byte for arithmetic.
- Unpacked (zoned) format is used for transmitting data to I/O devices.
- Pack and Unpack instructions are provided.

Decimal instructions are designed for operations requiring few computational steps between the source input and the documented output. Processing of this type is frequently found in commercial applications. Because of the limited number of arithmetic operations performed on each item of data, radix conversion from decimal to binary and back to decimal is not justified, and the use of registers for intermediate results yields no advantage over storage-to-storage processing. Hence, in the CE, decimal instructions are provided and both operands and results are located in main storage. Decimal instructions include addition, subtraction, multiplication, division, and comparison.

Decimal arithmetic operates on data in the packed format, in which two four-bit BCD digits are packed two to a byte. They appear in fields of variable length (from 1 to 16 bytes) and are accompanied by a sign in the rightmost four bits of the low-order byte. The use of packed digits within a byte and of variable-length fields within storage results in efficient use of storage and in increased arithmetic performance.

Decimal numbers may also appear in a zoned format for use with I/O devices operating in that format. The zoned format is not used in decimal arithmetic operations, but only for transmitting data to the I/O device. Instructions are provided for packing and unpacking decimal numbers so that they may be changed from the zoned (unpacked) to the packed format and vice versa.

Processing takes place right to left between main storage locations, except in the divide operation which is processed left to right. All decimal instructions use the two-address SS format. Each address specifies the leftmost byte of an operand. Associated with this address is a length field, indicating the number of additional bytes that the operand extends beyond the first byte.

Number Representation

Numbers are represented as right-aligned true integers with a plus or minus sign. Decimal digits 0–9 are represented in the four-bit BCD form by 0000 through 1001, respectively.

Codes 1010–1111 (10–15) are not valid as digits and are reserved for sign codes: 1010, 1100, 1110 and 1111 represent a plus; 1011 and 1101 represent a minus.

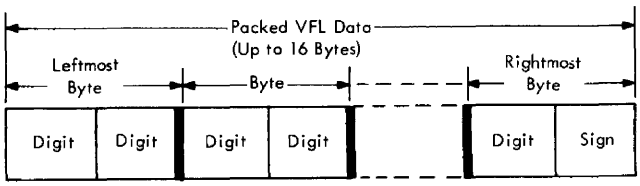
Digit	Code	Sign	Code
0	0000	+	1010
1	0001	–	1011
2	0010	+	1100
3	0011	–	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

All valid sign codes are recognized in decimal operations; however, the appropriate sign codes (and zone codes for the Unpack instruction) generated during the operation depend on the character set specified by PSW(12). If PSW(12) = 0, EBCDIC is selected, and code 1100 is generated for a plus sign, code 1101 is generated for a minus sign, and code 1111 is generated for a zone. If PSW(12) = 1, USASCII-8 is selected, and code 1010 is generated for a plus sign, code 1011 is generated for a minus sign, and code 0101 is generated for a zone.

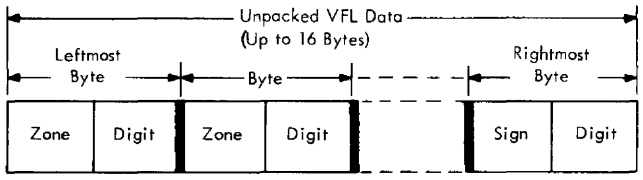
Formats

Decimal operands reside in main storage only. The operand field length may range from a minimum of one byte to a maximum of 16 bytes. The operands need not occupy the entire field length but are always right-aligned in the field; i.e., the sign of the operand is always in the rightmost byte of the specified field. This rightmost byte contains the lowest-order operand digit and the operand sign. All decimal instructions (except Divide) process the operands from low order to high order, or from right to left between main storage locations.

Data may be in the packed or unpacked (zoned) format. In the packed format, two four-bit BCD digits are placed adjacently in an eight-bit byte, except for the rightmost (low-order) byte of the field. In the low-order byte, a four-bit sign (sign of the decimal number) is placed to the right of the decimal digit.



In the unpacked or zoned format, a decimal digit normally occupies the four low-order bits of a byte, the numeric. The four high-order bits of a byte are called the zone. An exception is the rightmost byte in the field, where the sign of the decimal number occupies the zone position.



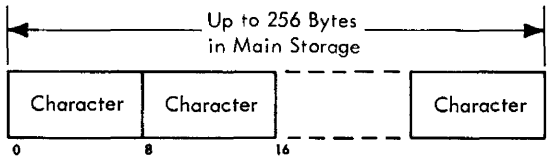
Logical Data

- Data is fixed-length or VFL.
- One byte of immediate data is held in some instruction formats.

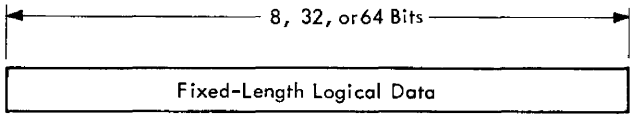
The logical instructions provide for moving, comparing, bit testing, bit connecting, translating, editing, and shifting operations. Except for the editing instructions, data is not treated as numbers. Editing converts packed decimal digits into alphanumeric characters; the digits, signs, and zones are recognized and generated as for decimal instructions.

Data resides in main storage or in LS, or is contained in the instruction format. The data may be a single byte, a word, a doubleword, or variable in length. When two operands participate in the operation, they have equal length, except in the editing instructions. The data format depends on the type of operation performed:

1. In storage-to-storage operations, data has a VFL format, starting at any byte address and continuing for a maximum of 256 bytes; it is processed left to right.



2. In storage-to-register operations, the main storage data may be either a word or a byte. The word must be located on a word boundary; that is, the low-order two bits of its address must be 0's. Data in GPR's normally occupies all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of the register participate, leaving the remaining 24 bits unchanged. In some shift operations, 64 bits of an even/odd pair of GPR's participate.



3. In operations which introduce data directly from the SI format instruction as an immediate operand, data is restricted to an eight-bit byte. Only one byte may be introduced per instruction, and only one byte from main storage takes part in the operation.

INDEX