IBM Field Engineering

Manual of Instruction

2075 Processing Unit -- Volume 3

Fixed Point

I Execute

Branch

Floating Point

Variable Field Length

223-2874-1

Form 223-2874-1
FES S26-7035

PREFACE

This is one of six Field Engineering manuals for the 2075 Processing Unit. These six manuals contain the unit theory of operation, reference diagrams to be used when troubleshooting, and maintenance procedures.

A basic knowledge of the IBM System/360 as contained in the IBM System/360 Principles of Operation, Form A22-6821 is considered a prerequisite for studying the unit theory of operation. The theory of operation is contained in a four volume manual identified as a Field Engineering Manual of Instruction (FEMI). Volume 1 is a prerequisite for the detailed information contained in volumes 2, 3, and 4. Volume 1 contains the introduction to the system and the processing unit and a description of the functional units (registers, adders, and decoders) of the processing unit. Volumes 2 and 3 contain detailed instruction analysis, and volume 4 contains detailed information on special features and power supplies and control.

The four volumes of theory of operation contain many references to the diagrams packaged in the associated Field Engineering Diagrams Manual (FEDM). All diagrams in the FEDM are identified by a four digit figure number and unless otherwise specified, all four digit figure references in the FEMI indicate that the figure is contained in the associated FEDM.

The complete titles and form numbers of the six 2075 Field Engineering Manuals are:

- 2075 Processing Unit--Volume 1, Comprehensive Introduction, Functional Units, Field Engineering Manual of Instruction, Form 223-2872
- 2075 Processing Unit--Volume 2, Theory of Operation: Storage Bus Control; Instruction Preparation; FLT, Logout, MCW; Interrupts, Field Engineering Manual of Instruction, Form 223-2873
- 2075 Processing Unit--Volume 3, Theory of Operation: Fixed Point, I Execute, Branch, Floating Point, Variable Field Length, Field Engineering Manual of Instruction, Form 223-2874
- 2075 Processing Unit--Volume 4, Special Features, Power Supply and Control, Appendix, Field Engineering Manual of Instruction, Form 223-2875
- 2075 Processing Unit, Field Engineering Diagrams Manual, Form 223-2876
- 2075 Processing Unit, Field Engineering Maintenance Manual, Form 223-2880

MAJOR REVISION (January, 1966)
This edition, Form 223-2874-1 is a major revision of the previous edition, Form 223-2874-0. The major changes in this edition are the expansion of the introduction to the "Fixed-Point" section, and the complete revision of "Fixed-Point Divide." There are minor changes throughout.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Systems Development Division, Product Publications, Dept. 520, CPO Box 120, Kingston, N. Y., 12401

CONTENTS

ILLUSTRATIONS

ABBREVIATIONS

AA	Addressing Adder
Adj	Adjust
AD (or AV)	Decimal Adder
AE	Exponent Adder
AEOB	Exponent Adder Output Bus
AM	Main Adder or Fixed Input of Main Adder
AMOB	Main Adder Output Bus
AMTC	Main Adder True/Complement or True/ Complement Input of Main Adder
AOB	Adder Output Bus
AOE	AND - OR - Exclusive OR
AP	(Mnemonic) Add Decimal (SS)
ASC	American Standard Interchange Code
ASCII	American Standard Interchange Code
AV (or AD)	Decimal Adder
BALR	(Mnemonic) Branch and Link (BR)
BCD	Binary Coded Decimal
BCR	(Mnemonic) Branch on Condition (RR)
BCTR	(Mnemonic) Branch on Count (RR)
BCU	Bus Control Unit
Bin	Binary
BOP	Buffer Operation Register
CLC	(Mnemonic) Compare Logical (SS)
Comp	Complement
CP	(Mnemonic) Compare Decimal (SS)
CVD	(Mnemonic) Convert to Decimal (RX)
DB	Digit Buffer
DC	Digit Counter
DCR	Digit Counter Register
Dec	Decimal
DP	(Mnemonic) Divide Decimal (SS)
ED	(Mnemonic) Edit (SS)
EDMK	(Mnemonic) Edit and Mark (SS)
ELC	E (Unit) Last Cycle
EOP	E (Unit) Operation Register
ER	Exponent Register
FLOUT	Floating Point (Register) Out
FLP	Floating Point
GPR	General Purpose Register
GR	General (Purpose) Register
GROUT	General Register Out
GSR	Gate Select Register
Gt	Gate
HOD	High-Order Digit
HS	Half-Sum
Hwd Add	Halfword Add
Hwd Log	Halfword Logical
IC	Instruction Counter
ICR	Instruction Counter Register
IE	I Unit Execution
IOP	I Unit Operation Register
IS	Iteration Sequence
KBR	Key Buffer Register
LBG	Left Byte Gate
LC	Last Cycle
LCDR	(Mnemonic) Load Complement -- Long FLP (RR)
LCER	(Mnemonic) Load Complement -- Short FLP (RR)
LNDR	(Mnemonic) Load Negative -- Long FLP (RR)
LNER	(Mnemonic) Load Negative -- Short FLP (RR)
LOD	Low-Order Digit
LPDR	(Mnemonic) Load Positive -- Long FLP (RR)
LPER	(Mnemonic) Load Positive -- Short FLP (RR)
LTDR	(Mnemonic) Load and Test -- Long FLP (RR)
LTER	(Mnemonic) Load and Test -- Short FLP (RR)
Lth	Latch
MDR	(Mnemonic) Multiply -- Long FLP (RR)
ME	(Mnemonic) Multiply -- Short FLP (RX)
MER	(Mnemonic) Multiply -- Short FLP (RR)
MODAR	Modified Addressable Register
MVC	(Mnemonic) Move (SS)
MVN	(Mnemonic) Move Numerics (SS)
MVO	(Mnemonic) Move With Offset (SS)
MVZ	(Mnemonic) Move Zone (SS)
NC	(Mnemonic) AND (SS)
OC	(Mnemonic) OR (SS)
OPF	Operand Fetch
Par	Parity
PF	Prefetch
PH	Parity Adjusted for Removal of HOD
PK (or PACK)	(Mnemonic) Pack (SS)
PL	Parity Adjusted for Removal of LOD
PSW	Program Status Word
RBG	Right Byte Gate
RDD	(Mnemonic) Read Direct (SI)
RR	(Instruction Format) Both Operands from GPR's
RS	(Instruction Format) One Operand from a GPR, the other from storage
RX	(Instruction Format) One Operand from a GPR, the other from an indexed storage location
S	S (Pointer or Register)
SAR	Storage Address Register
SBI	Storage Bus In

SBO	Storage Bus Out
SC	Shift Counter
Sel	Select
Seq	Sequence
SF	Store-Fetch
SI	(Instruction Format) One Operand from storage, the other is immediate
SLA	(Mnemonic) Shift Left Single (RS)
SLDA	(Mnemonic) Shift Left Double (RS)
SP	(Menmonic) Subtract Decimal (SS)
SRA	(Mnemonic) Shift Right Single (RS)
SRDA	(Mnemonic) Shift Right Double (RS)
SS	(Instruction Format) Both Operands from storage
SU	Set-Up
T	T (Pointer or Register)
T/C	True/Complement
TD	T (Pointer) Decode
Term	Termination
TR	(Mnemonic) Translate (SS)
TRT	(Mnemonic) Translate and Test (SS)
UNPK	(Mnemonic) Unpack (SS)
VFL	Variable Field Length
WRD	(Mnemonic) Write Direct (SI)
XC	(Mnemonic) Exclusive OR (SS)
Y	Y (Length Counter)
Z	Z (Length Counter)
ZAP	(Mnemonic) Zero and Add (SS)
ZD	Z (Counter) Decode

INTRODUCTION

OPERANDS

- Are 16, 32, or 64 bits long.
- High-order bit is sign bit in numeric operands.
- Negative numbers are in 2's complement form.

Most operands are 32 bits long, with the high-order bit used as the sign of numeric operands (1 is minus, 0 is plus). Some instructions, however, make use of halfword operands (16 bits), and some, double word operands (64 bits). The high-order bit in numeric operands is considered a sign bit, while for logical operands the high-order bit is just another data bit.

Halfword operands are expanded to a full word after their delivery from core storage. The expansion of the halfword is achieved by propagating (extending to the left) the sign bit, thereby leaving the value of the operand unaffected. Thereafter, the expanded halfword is handled as any full word. For example, after expansion, the add halfword is executed the same as add.

Double word operands, used in divide and some shift operations, are contained in an even-odd pair of general registers. The instructions using double operands must specify the even general register to avoid a specification interrupt.

Numeric Operands

Unlike many machines, negative numbers are held in 2's complement form, with a 1 in the sign (high-order) bit. The 2's complement of a number is obtained by inverting all bits and adding a 1 to the low-order bit. For example, the number 6_{16} is represented by

```
8 4 2 1
0 1 1 0
```

when positive, and by

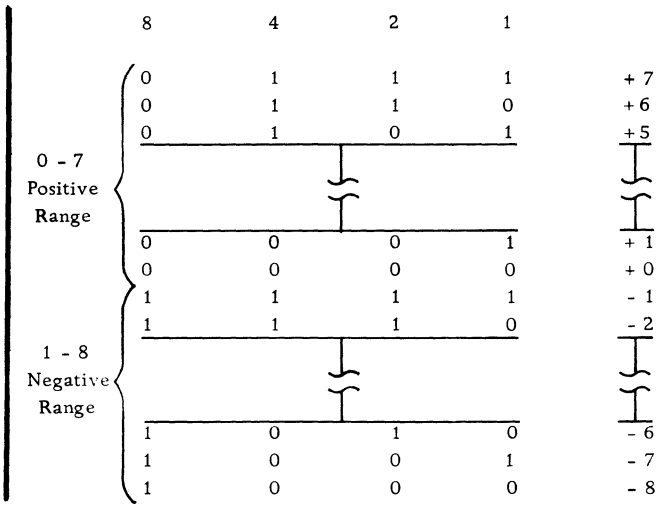
```
8 4 2 1
1 0 1 0
```

when negative. Note that the hexadecimal value of the second binary group (10_{16}) is the 16's complement of the first binary group (6_{16}). Therefore, any binary number can be broken into four-bit groups and be represented by hexadecimal digits; the 16's complement of those hexadecimal digits will then be equivalent to the 2's complement of the binary number.

Keeping negative numbers in complement form simplifies processing. When an operation, such as add, yields a negative (complement) result, the result is stored in the general registers without alteration. In many machines, where negative numbers are represented in true form, an additional pass through the adder must be taken to make a complement (negative) result true. By leaving the result in complement form, the System/360 Model 75 saves the cycle required to make a pass through its adder.

Numbers Range

For any given field size, the complement or negative numbers are equal in quantity to the positive numbers. However, a negative zero does not exist, but a positive zero does; and, the maximum negative number (one followed by all zeros) is greater in absolute value than the maximum positive number (zero, followed by all ones). This is illustrated in the following example, where the field is only 4 bits in length:



OVERFLOW

- Caused by lost significant bits.
- Sets CC to 3 and causes fixed-point overflow interrupt when the fixed-point overflow interrupt mask is on.

When an operation either produces a result that is greater than the machine's capacity, or loses bits that should be saved, a fixed-point overflow occurs. The result, while invalid, is placed in the general

registers where results are stored. The condition code is set to 3, and an interrupt occurs if the fixed-point overflow mask bit is 1.

In the preceding example, the representable range for positive numbers is 0 to 15, and for negative numbers is 1 to 16. Therefore, any operation that attempts to produce a result of greater than +15, or less than -16, signifies an overflow.

In load complement, which loads the complement of one general register into another, an overflow occurs if the operand is the maximum negative number. Using the four numeric bit machine, it can be seen that if -16 (10000) were complemented (subtracted from zero), the result is +16, out of range. However, the bit arrangement of the result (10000) is the same as the original operand.

CONDITION CODE

- Set by most fixed-point instructions.
- Can be tested by branch-on-condition instructions.

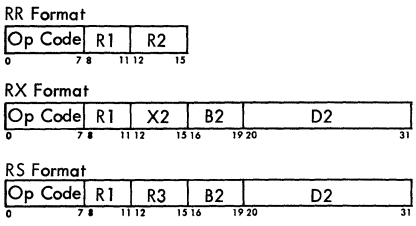
Most fixed-point instructions set the condition register (PSW bits 34 and 35) during the last cycle of their execution. The four states of the condition register are used, by various instructions, to indicate the relation of an operand to zero, of one operand to another, or overflow. The condition register can be used for decision-making by branch-on-condition instructions.

For some instructions, for example, add and add logical, the setting of the condition register is the only difference in their execution. Different interpretations are given to the four states of the condition register for these two instructions.

INSTRUCTION FORMAT

- Fixed-point/fixed-sequence instructions have RR, RX, and RS formats.

Fixed-point/fixed-sequence instructions use the following three formats:



In these formats, R1 specifies the address of the general register that contains the first operand. The second operand location, if any, is defined differently for each format.

In the RR format, the R2 field specifies the address of the general register that contains the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general registers that are specified by the X2 and B2 fields are added to the content of the D2 field to form an address that designates storage location of the second operand.

In the RS format, the content of the general register that is specified by the B2 field is added to the content of the D2 field to form an address. This address designates the storage location of the second operand in load multiple and store multiple. In the shift operations, the address specifies the amount of shift. The R3 field specifies the address of a general register in load multiple and store multiple and is ignored in the shift operations.

A zero in a X2 or B2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register for both address modification and operand location. Address modification is always completed before operation execution.

The contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged, except for the storing of the final result.

PROGRAM INTERRUPTS

- Program interrupts are caused by programming errors and will block, terminate, or suppress the instruction in error.

Program interrupts are caused by various programming errors, some of which are detected during instruction preparation time (T1 and T2); others are detected during instruction execution time. The errors detected during T1 and T2 are the specification errors. These prevent the start of any execution unit, causing an interrupt to be taken instead. Of the errors detected during the execution time of an instruction, some end the instruction prematurely, others nullify it. Fixed-point overflow does not interfere with execution of the instruction. These errors and their effect are described under "Theory of Operation."

THEORY OF OPERATION

The E unit is started when the preparation of the fixed-point instruction is completed, the E and IE units are not busy with the previous instruction, and there is no interrupt condition. This point in an instruction is commonly called "I to E transfer" which is a signal that combines with "no interrupt" to turn on the E busy or the IE busy trigger (Figure 5276).

At the A pulse coincident with I to E transfer, the first FXP trigger is turned on and is followed by its latch. In RR instructions, the first FXP latch is turned on immediately following the trigger; in RX instructions the first FXP latch is held up, keeping the E unit idle, until the J register is loaded with valid data (Figure 5404). If invalid data is received, as when an illegal fetch address is used, the first FXP latch turn-on is blocked and the ELC trigger is turned on to end the instruction (see "Operand Store-Fetch Errors").

The I unit delivers the required register operands to the E unit, makes all fetch requests and most store requests. At I to E transfer the required register operands have, for most fixed-point instructions, been delivered to the M register through RBL, and the accept pulse from BCU has been received for fetched operands (See Figure 1).

Register Operands

Register operands R1 and R2 are gated into RBL during T2 for RR instructions even though one of these operands may not be required (see Figure 6400). In RX instructions, the R1 operand is sent to RBL, and so is R1 + 1 if R1 is even. As in the RR instructions, the R1 or R1 +1 operand may not be required, and if not, is ignored by the E unit. R1 is delivered to RBL left; R2 and R1 +1 are delivered to RBL left; R2 and R1 + 1 are delivered to RBL right.

The gating for GR to RBL transfer occurs during T2. However, some instructions require their operand deliveries during one or more execution cycles. If so, the general register out (GROUT) trigger is turned on at I to E transfer to accomplish the gating and stays on as long as required.

Put-Away

Put-away refers to storing an instruction result in K0-31 in the general register specified by ER1 (Figure 5401). For instructions requiring a single put-away, the transfer is made in ELC. For instructions requiring a double put-away (for instance, double word shift instructions), the transfers are made in both the PA and ELC cycles.

The controlling trigger for the K to GR transfer is the release cycle trigger (Figure 5402). This trigger is turned on at the late B pulse preceding the put-away(s) and is turned off in ELC (with a late B pulse). Therefore, the release cycle trigger straddles the clock pulse (early B) that sets the general registers. Instructions not requiring a put-away and the error conditions that must block it turn on the block PA trigger (Figure 5402). This trigger, or its turn-on condition, prevents setting the release cycle trigger.

Condition Register Setting

The condition register (PSW bits 34 and 35) is set at the A clock pulse following the ELC cycle for the instructions that require it. Some instructions, like Store, do not set the condition register. The output of circuits comparing results to zero, or one operand to another, for example, set latches in the E unit which correspond to PSW positions 34 and 35 (see Figure 5403). These E unit latches are then used to set PSW positions 34 and 35 with the gate "E Set CR," which is coincident with the ELC latch.

The condition register settings and their significance to applicable instructions are shown with the instruction flow diagrams in the FE Diagrams Manual, 2075 Processing Unit, Form 223-2876.

Program Interrupts

E time program interrupts possible with the fixed-point instructions follow.

Operand Store-Fetch Errors

Four types of illegal addressing are connected with fetching or storing operands:

1. Invalid fetch address
2. Storage address protect (SAP) fetch error
3. Invalid store address
4. Storage address protect (SAP) store error

Invalid Fetch Address: An out-of-range storage address used to fetch an operand. In this operation, CPU still receives an advance pulse, but also receives an error signal to turn on the address invalid trigger (Figure 5404). The J register receives, instead of the SBO output, the contents of the panel keys (with good parity).

The invalid address trigger prevents the E unit from executing the instruction by forcing ELC and

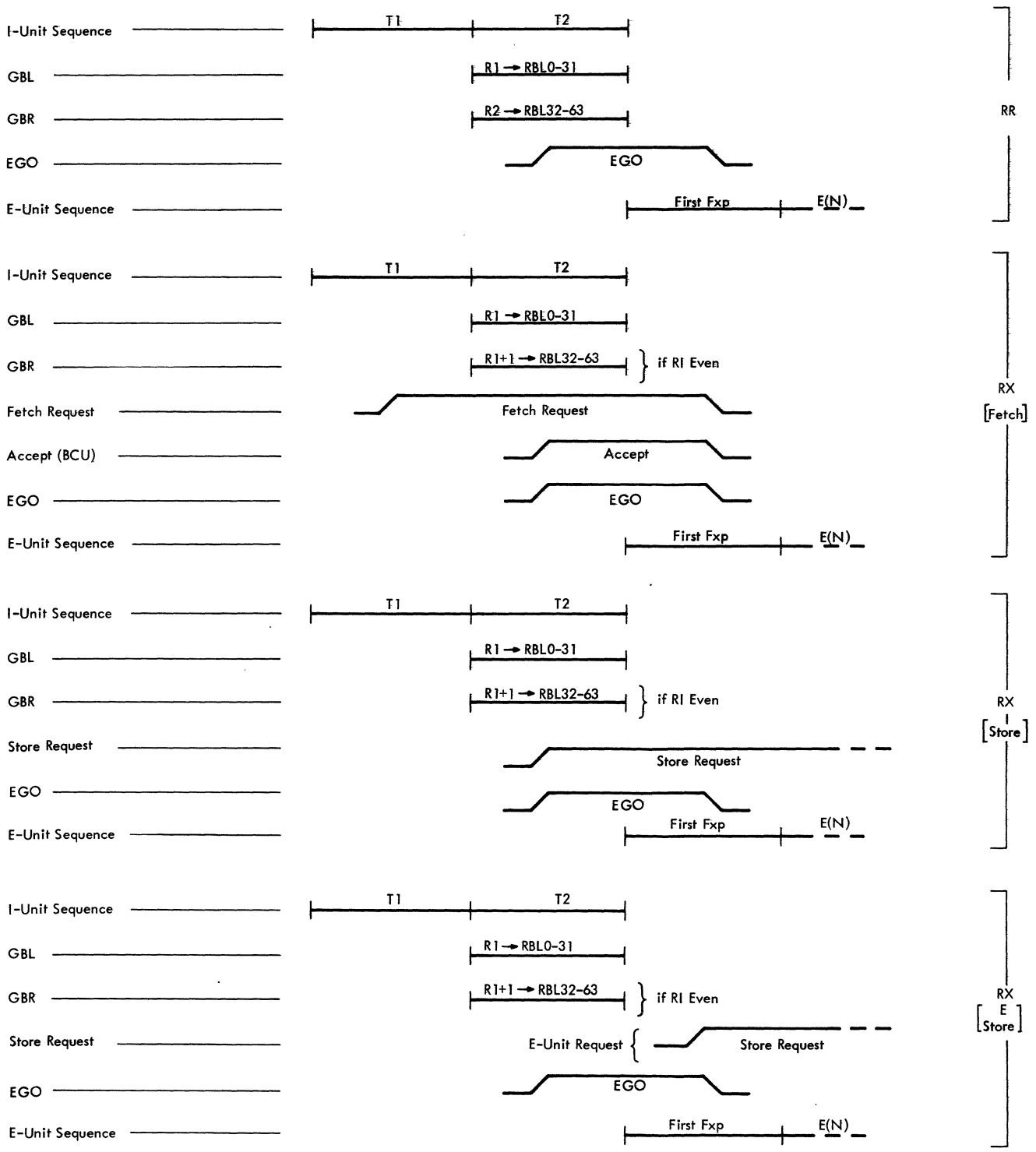


FIGURE 1. I-E TRANSFER

blocking the turn-on of the first FXP latch (the first FXP trigger is on since I to E transfer). Also, the address invalid trigger causes turn on of: (1) the block PA trigger to prevent changing the general registers; (2) the E interrupt trigger to cause a program interrupt; and (3) the address interrupt trigger to identify the addressing interrupt.

Storage Address Protect (SAP) Fetch Error: Occurs when the storage protection key (PSW bits 8-11) is not zero, does not match the SPF key, and the read-protect bit in the SPF unit is 1. A SAP check signal is sent to the CPU and, with the advance pulse, turns on a SAP check trigger (Figure 5404). As with an invalid fetch address, J is loaded from the panel keys (with good parity) instead of from the SBO.

As with the invalid address trigger, the SAP check trigger (identified with the J register) prevents the E unit from executing and instruction using the illegal address by forcing ELC and blocking the turn on of the first FXP latch (the first FXP trigger is on since I to E transfer). Also, the SAP check trigger causes turn on of: (1) the block PA trigger to prevent changing the general registers, (2) the E interrupt trigger to cause a program interrupt; and (3) the SAP interrupt trigger to identify the SAP fetch interrupt.

Invalid Store Address: An out-of-range storage address used in a store operation. The operand never reaches its destination, but the accept pulse is sent to the CPU as though a legitimate address were used, and the instruction is allowed to complete as it would normally. Because of the invalid address, however, the BCU sets one of its triggers called "invalid store buffer" (Figure 5204) whose output is sent to the CPU's interrupt detection circuits (Figure 5350). At ELC of the instruction using the invalid store address, the interrupt detection circuits recognize the error and start an interrupt sequence.

Storage Address Protect (SAP) Store Error: Occurs when an instruction attempts to store a word in an area of storage whose SPF protection key is not matched by the protection key in the PSW (when the PSW key is other than zero). The operand never reaches its destination, but the accept pulse from BCU is sent to the CPU as though a legitimate address were used. Because of the SAP error, however, the BCU sets a trigger within it called "CPU SAP" (Figure 5204) whose output is sent to the CPU's interrupt detection circuits (Figure 5350).

Because of its late arrival in CPU, the signal generated by the CPU SAP trigger might not cause a store SAP interrupt immediately following the instruction using the illegal address. The interrupt may be taken after the next instruction, or if a large capacity storage (LCS) is involved, the interrupt may not be taken until many instructions later because of the LCS's relatively slow response (see Interrupt Examples in "Interrupts," FEMI, 2075 Processing Unit, Volume 2, Form 223-2873).

Fixed-Point Overflow

When the load positive, load negative, or algebraic add or subtract instructions produce a result greater than 32 bits, or when the algebraic left shift instructions lose significant high-order bits, the instruction is completed normally and the condition code is set to 3. If the fixed-point overflow mask bit (PSW 36) is 1, the E interrupt trigger and fixed-point overflow trigger are turned on and an interrupt is taken at the end of the instruction.

Fixed-Point Divide

When the quotient exceeds 32 bits in a divide operation, including division by zero, the fixed-point interrupt and E interrupt triggers are turned on and an interrupt is taken at the end of the instruction. Put-aways are blocked by turning on the block PA trigger (Figure 5402).

The divide instruction may be cut short depending on when the error is detected.

MODAR Trigger

The modified addressable register (MODAR) trigger is associated with the retry-no-retry feature, which enables a decision to be made as to whether an instruction that caused a machine check can be retried; that is, whether that instruction has changed a register. A register in this case refers to either a general register, floating-point register, PSW, or storage location. If the instruction did change a register, before the machine check, it is not retryable because it may have changed its own operand.

When a machine check occurs and the instruction has already changed a register, that instruction, as indicated by the MODAR trigger being on, is not retryable. If, when the machine check occurs, the instruction has not progressed far enough to change any register, the MODAR trigger is not turned on, indicating that the instruction causing the machine check is retryable.

The MODAR trigger is one of the 1216 triggers logged out in a machine check, and therefore, can be interrogated by an examining program.

LOAD (L, LR)

- Operand 2 is placed in the operand 1 location.

The load instructions are executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of execution, the second operand is located as:

Instruction Format	Operand 2
RR	M32-63
RX-even	J0-31
RX-odd	J32-63

Figure Reference: Figure 6300

First FXP

The second operand, from the M register (RR) or J register (RX), is sent through the adder to the left half of the K register.

ELC

A put-away is made from the K register to the general register specified by ER1, and the MODAR trigger is set.

LOAD AND TEST (LTR)

- Operand 2 is placed in the operand 1 location.
- CC is set to record the relation of the result to 0.

The load and test instruction is executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of execution, the R2 operand is located in M32-63.

Figure Reference: Figure 6300

First FXP

The R2 operand, in the M register, is gated through the adder to the left half of the K register.

ELC

A put-away is made from the K register to the general register specified by ER1. MODAR is set, and 0, 1, or 2 is gated to the condition register.

The setting of the condition register is determined by examining the K register 0 bit (sign bit) and the K0-63 zero latch.

LOAD POSITIVE (LPR)

- Operand 2, made positive if it is negative, is placed in the operand 1 location.
- CC is set to record the relation of the result to 0, or to record overflow.
- When the R2 operand is the maximum negative value (100---00), overflow occurs and the number remains unaltered.

The load positive instruction is executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of the execution, the R2 operand is located in M32-63.

Figure Reference: Figure 6300.

First FXP

The R2 operand, in the M register, is gated through the adder to the left half of the K register. If the R2 operand is negative, the AM complement and hot 1 triggers are turned on to complement the operand as it passes through the adder.

ELC

A put-away is made from the K register to the general register specified by ER1. MODAR is set, and a 0, 2, or 3 is gated to the condition register.

The setting of the condition register is determined by examing the K0 bit (sign bit), the K0-63 zero latch, and the carries from adder positions 0 and 1.

LOAD NEGATIVE (LNR)

- Operand 2, made negative if it is positive, is placed in the operand 1 location.
- CC is set to record the relation of the result to 0.
- When the number 0 is complemented, the result is 0.

The load negative instruction is executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of execution, the R2 operand is located in M32-63.

Figure Reference: Figure 6300.

First FXP

The R2 operand, in the M register, is gated through the adder to the left half of the K register. If the R2 operand is positive, the AM complement and hot 1 triggers are turned on to complement the operand as it passes through the adder. The result of 2's complementing a 0 operand is 0.

ELC

A put-away is made from the K register to the general register specified by ER1. MODAR is set, and a 0 or 1 is gated to the condition register.

The setting of the condition register is determined by examining the K0 bit (sign bit) and the K0-63 zero latch.

LOAD COMPLEMENT (LCR)

- The complement of operand 2 is placed in the operand 1 location.
- CC is set to record the relation of the result to 0, or to record overflow.
- If the maximum negative value (100---000) is complemented, overflow is recorded in the condition code and an interrupt is taken if the fixed-point overflow mask bit (PSW 36) is one.

The load complement instruction is executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of the execution, the R2 operand is located in M32-63.

Figure Reference: Figure 6300.

First FXP

The R2 operand, in the M register, is gated through the adder to the left half of the K register. The AM complement and hot 1 triggers are turned on to complement the operand as it passes through the adder. Complementing does not alter a zero operand or the maximum negative value (100--00).

ELC

A put-away is made from the K register to the general register specified by ER1. MODAR is set, and a 0, 1, 2, or 3 is gated to the condition register.

The setting of the condition register is determined by examining the K0-63 zero latch, the K0 bit (sign bit) and the carries from adder positions 0 and 1.

LOAD ADDRESS (LA)

- The 24-bit address formed by X2, B2, and D2 is placed in the 24 low-order positions of the general register specified by R1; bits 0-7 of GR R1 are made 0.
- No operand fetch is made and the CC is not set.

The load address instruction is executed in three cycles, defined by first FXP, Hwd Log, and ELC triggers. At T1, the address calculation is made from the X2, B2, and D2 fields, as for any RX instruction. At TN T2, the calculated address is placed in SAR and H registers, but no fetch request is made. Block ICM is turned on at the beginning of T2 to prevent the ICR from being gated into the incrementer, which will be used in the second execution cycle. Block T1 is also generated, to prevent the next TN T2 pulse from altering the H register. At I to E transfer and an A pulse, the first FXP trigger is turned on.

Figure Reference: Figure 6301.

First FXP

The turn-off of block T1M is gated to allow T1 in the next cycle. The H register cannot change until after its contents (X2 + B2 + D2) have been used. No data flow occurs in this cycle in order to leave the incrementer, used as a data path in load address, free for a high-order advance of the ICR.

Hwd Log

The H register is gated to the incrementer, and the incrementer with its extender is gated to the K register. Therefore, positions 0-7 of the K register receive zeros, while positions 8-31 receive the 24-bit address.

ELC

A put-away is made from the K register to the general register specified by ER1, and the MODAR trigger is set.

ADD (A, AR)

- Operand 2 is added to operand 1, and the algebraic sum is placed in the operand 1 location.
- CC is set to record the relation of the result to 0, or to record overflow.

Overflow is possible and causes the sign of the result to be opposite that of the two numbers added; the magnitude of the result is also invalid. The overflow is recorded in the condition code and, if the fixed-point overflow mask bit (PSW 36) is 1, a program interrupt takes place.

The add instructions are executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of execution, the two operands are located as:

Instruction Format	Operand 1	Operand 2
RR	M0-31	M32-63
RX-even	M0-31	J0-31
RX-odd	M0-31	J32-63

Figure Reference: Figure 6302.

In computers that keep negative numbers in true (not complement) form, an examination of the signs of the two operands must be made to see whether the result should be their sum or difference. If the result should be their difference, one of the operands is complemented before being added to the other. In the System/360, the sign examination is unnecessary because negative numbers are kept in complement (2's) form. Adding a negative and a positive number, therefore, automatically yields their difference (actually their algebraic sum -- for example, +3 and -5 equals -2, the 2 being the difference in absolute values, but -2 being the algebraic sum of the two numbers); adding two negative or two positive numbers yields their sum.

First FXP

Operand 1 is gated from the M register to the left side of the adder; operand 2 is gated from the M register (RR) or J register (RX) to the right side of the adder. The result is gated to the left half of the K register.

ELC

A put-away is made from the K register to the general register specified by ER1. MODAR is set, and a 0, 1, 2, or 3 is gated to the condition register.

The setting of the condition register is determined by examining the K register 0 bit (sign bit), the K0-63 zero latch, and the carries from adder positions 0 and 1.

SUBTRACT (S, SR)

- Operand 2 is subtracted from operand 1, and the algebraic difference is placed in the operand 1 location.
- CC is set to record the relation of the result to 0, or to record overflow.
- Overflow occurs when the magnitude of the difference exceeds 31 bits; if the fixed-point overflow mask bit (PSW 36) is 1, an interrupt is taken.

Execution of the subtract instructions is similar to the execution of the add instructions, with one exception. As the operands are gated through the adder, the AM complement and hot 1 triggers are unconditionally turned on to complement operand 2. The initial location of operands and the setting of the condition register are also the same as the add instructions.

Complementing operand 2 adheres to the basic rule of algebra of changing the sign of the subtrahend (number being subtracted) and adding. The result is always the algebraic difference of the two numbers. This is less complicated than the computers that keep negative numbers in true form and must decide on the basis of operand signs whether to complement, and later, whether to recomplement to make a true result.

Figure Reference: Figure 6302.

CONDITION CODE SETTING FOR FIXED-POINT LOAD-TYPE AND ALGEBRAIC ADD-SUBTRACT INSTRUCTIONS

- CC is set to record the relation of the result to 0, or to record overflow.
- Sign of result, K zero indication, and high-order carries from the adder are compared.

The condition code is used by the fixed-point load-type and algebraic add-subtract instructions to compare the results of their operations to zero. That is, to record whether the result operands are equal to, less than, or greater than, zero. Specifically, these instructions are:

Load and test
Load positive
Load negative
Load complement
Add
Add halfword
Subtract
Subtract halfword

The condition code is also used to indicate the presence of overflow for all of the above codes except load and test and load negative; overflow is not possible for these two instructions. The condition code (or condition register -- PSW bits 34 and 35)

settings and their interpretation for the above codes are:

CC Bits 34 and 35	Interpretations
0 0	Result is zero
0 1	Result is less than zero
1 0	Result is greater than zero
1 1	Overflow

Condition Code 0, 1, and 2

Determining the relation of a result to zero is done by examining the K register 0 bit (sign bit) and the K register zero latch during ELC. If the K register zero latch is on, the result is zero and CR bits 34 and 35 are unaffected. If the K0 bit is off and the K zero latch is off, the result is greater than zero (positive), and CR bit 34 is set. If the K0 bit is on, the result is less than zero (negative) and CR bit 35 is set.

The bits generated by these examinations are first set into the CR 34 and CR 35 latches (KX 621) in the E unit. At A clock following the ELC cycle, the bits are transferred to the corresponding PSW bits 34 and 35 with the gate E set CR (KX 625).

Condition Code 3 (Overflow)

Overflow occurs when the result of an operation is greater than 31 bits (without sign). Overflow is detected by comparing the carries out of positions 0 and 1 from the main adder. If a carry from both positions is present, or absent, the result is within the representable range. If a carry occurs from one position and not the other, overflow exists, and a 3 is set into the condition code. If the PSW 36 bit (fixed-point overflow mask bit) is one, an interrupt is started as well.

In load positive and load complement, overflow occurs when the maximum negative value (100---000) is complemented.

Except for the maximum negative value, overflow changes the sign of the result to opposite of what it should be. For examples of operands that cause overflow and the high-order carries they produce, see the section entitled "Overflow on Algebraic Add-Subtract Instructions."

ADD LOGICAL (AL, ALR)

- The logical operand 2 is added to the logical operand 1 and the sum is placed in the operand 1 location.
- CC is set according to zero/non-zero, and carry/no-carry produced by the result.

The high-order bits of both operands are treated as data and not sign bits; the operands are therefore classified as logical. The logical add instructions differ from their add instruction counterparts in the setting of the condition code and in the absence of overflow.

The initial operand locations and execution of the add logical instructions (RR and RX) are identical to the corresponding add instructions. Only the setting of the condition code differs. The state of the K0-63 zero latch and the C out of AM 0 trigger-latch determine the setting of the condition code. The logic for setting CR 34 (2 bit) is shown on Systems KX 641; the logic for setting CR 35 (1 bit) is shown on Systems KX 655.

Figure Reference: Figure 6302.

SUBTRACT LOGICAL (SL, SLR)

- The logical operand 2 is subtracted from the logical operand 1 and the difference is placed in the operand 1 location.
- CC is set according to zero/non-zero, and carry/no-carry produced by the result.

The high-order bits of both operands are treated as data and not sign bits; the operands are therefore classified as logical. The logical subtract instructions differ from their subtract instruction counterparts in the setting of the condition code and in the absence of overflow.

The initial operand locations and execution of the subtract logical instructions (RR and RX) are identical to the corresponding subtract instructions, which complement operand 2 as it passes through the adder. Only the setting of the condition code differs. The state of the K0-63 zero latch and the C out of AM 0 trigger-latch determine the setting of the condition code. The logic for setting CR 34 (2 bit) is shown on Systems KX 641; the logic for setting CR 35 (1 bit) is shown on Systems KX 655.

Figure Reference: Figure 6302.

COMPARE (C, CR)

- Operand 1 is compared to operand 2 and the CC is set to record their relation.
- Execution is identical to the subtract instructions with the absence of a put-away.

The compare instructions are executed in two cycles, defined by the first FXP trigger and the ELC trigger. At the start of execution, the two operands are located as follows:

Instruction Format	Operand 1	Operand 2
RR	M0-31	M32-63
RX-even	M0-31	J0-31
RX-odd	M0-31	J32-63

Figure Reference: Figure 6303.

First FXP

Operand 1 is gated from the M register to the left side of the adder; operand 2 is gated from the M register (RR) or J register (RX) to the right side of the adder. The AM complement and hot 1 gates are turned on to complement operand 2 as it passes through the adder. The output of the adder is sent to the K register for a zero check; this is part of the information required to determine the setting of the condition code.

ELC

A put-away to a general register is prevented by turning on the block PA trigger. The MODAR trigger is set, and a 0, 1, or 2 is gated to the condition register.

The condition register inputs are determined by the operand signs, the carry from adder output 0 position, and by the state of the K0-63 zero latch. The sign indications are obtained from the R1 and R2 sign triggers which are set by the operand sign positions during first FXP (see Figure 5400). The interpretation given the above indications for setting the condition register is explained in the following text.

Compare Logic

Comparing the relative values of the two algebraic operands is accomplished by examining the carry/no-carry out of the adder position 0 (the sign position), and the zero indication from the K register, where the adder output is received. The expressions used in the comparisons, and their significance, are:

Systems KX 641 (Signs Unlike) $\overline{(C\ Out\ 0)}$ + (Signs Alike) (C Out 0) $\overline{(K0\text{-}63=0)}$ = Operand 1 High = CC34

Systems KX 661 (Signs Unlike) (C Out 0) + (Signs Alike) $\overline{(C\ Out\ 0)}$ = Operand 1 Low = CC35

The first half of each expression (Signs Unlike) $\overline{(C\ Out\ 0)}$ and (Signs Unlike) (C Out 0), deals with the operand sign position only (see Figure 2). When signs are unlike, the carry/no-carry from position 0 distinguishes between the positive and negative operands. This is possible because after complementing, both operands are positive or both are negative. A carry from position 0, therefore, is impossible with positive signs and indicates that operand 2 (the complemented operand) is negative. A carry from position 0 always occurs with negative signs, and indicates that operand 2 was positive before complement.

The second half of each expression (Signs Alike) (C Out 0) $\overline{(K0\text{-}63=0)}$ and (Signs Alike) $\overline{(C\ Out\ 0)}$, is a test of the magnitude (see Figure 2). When signs are alike, the carry/no-carry from position 0 and the K zero indication provide enough information to tell when operand 1 is higher, lower, or equal to operand 2.

When operands 1 and 2 are equal, none of the expressions above is satisfied (see Figure 2). Neither of the condition code triggers is set, therefore, indicating an equal condition.

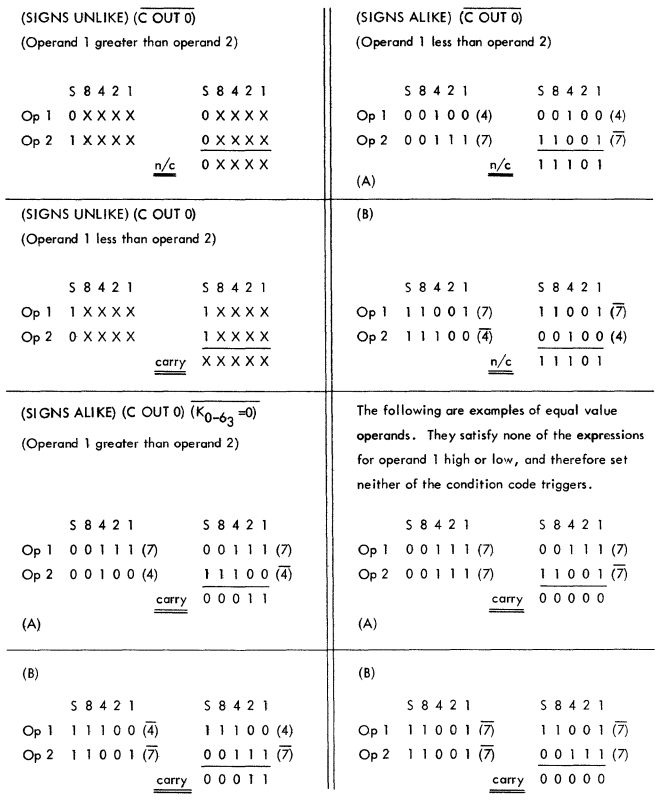


FIGURE 2. COMPARE EXAMPLES

COMPARE LOGICAL (CL, CLR)

- The logical operand 1 is compared to the logical operand 2 and the CC is set to record their relation.

Execution of the compare logical instructions is identical to the execution of the corresponding compare instructions (C, CR). Only the setting of the condition register differs. The compare instructions treat the operands as 31-bit signed integers while the compare logical instructions treat the operands as 32-bit unsigned integers. The logic used to set the condition register for compare logical is shown in the following expressions:

Systems KX 645 $\overline{\text{K0-63 Zero Lth}}$	C Out AM 0 Tgr Lth = CR 34
Systems KX 661	C Out AM 0 Tgr Lth = CR 35

Examples:

	Op 1 Greater	Op 1 = Op 2	Op 1 Smaller
	0 1 2 3 4	0 1 2 3 4	0 1 2 3 4
Operand 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1
Operand 2	0 0 0 0 0	0 0 0 0 1	0 0 0 1 0
Operand 2	1 1 1 1 1	1 1 1 1 0	1 1 1 0 1
	1	1	1
	0 0 0 0 1	0 0 0 0 0	1 1 1 1 1
	carry	carry	n/c

Figure Reference: Figure 6303.

STORE (ST)

- Operand 1 is stored in the operand 2 location.

The store instruction takes two cycles to execute if storage is immediately available to CPU, and more than two cycles if it is not. The I unit makes the store request at I to E transfer. If the requested storage is not busy, the accept pulse is received in the following cycle (first FXP) to turn on ELC. If the accept pulse is delayed, store (idle) cycles are taken between first FXP and ELC until the accept pulse arrives from BCU. At the start of execution, the R1 operand is located in M0-31.

Figure Reference: Figure 6304.

First FXP

The R1 operand in the M register is gated through the adder to the K register. The first first FXP cycle is effective (allows the adder output to enter the K register) because the first FXP latch is turned on immediately following the T2 cycle.

To prevent a put-away, the block PA trigger is turned on at the A pulse that follows this cycle, and stays on through the ELC cycle.

Store

The store trigger is turned on after the first FXP trigger to define idle cycles while the E unit waits for the accept pulse from BCU. The store trigger remains on for as long as the delay exists, and stays on through the ELC cycle. If there is no delay, the store trigger is on during ELC only. The MODAR trigger is turned on by the store cycle(s).

ELC

The ELC trigger is turned on at A clock following the receipt of the accept pulse (which can arrive as early as the first FXP). At EB time of ELC, the K register is set into the SBI latches.

The word (left or right) selected for storage input is determined by the mark register. The mark register is set for the left four or the right four bytes of a double word at I to E transfer on the basis of H bit 21 being either a 0 (left four bytes) or a 1 (right four bytes).

A general register put-away is prevented by the block PA trigger, which is turned on at A pulse following the first FXP cycle. T2 is allowed to be on during this cycle because the accept pulse removed the block in the previous cycle (E TF block T2M on accept -- KX 515).

HALFWORD EXPANSION

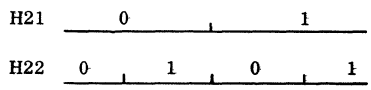
- The location of the halfword (16 bits) in the J register is specified by the H register bits 21 and 22.
- Without altering their value, halfword operands are expanded to a full word before being used.
- Two cycles are taken for this expansion, defined by the first FXP and the Hwd Log triggers.

Halfword operands (two bytes, or 16 bits) are specified by the following RX fixed-point instructions:

Load halfword
Add halfword
Subtract halfword
Compare halfword
Store halfword
Multiply halfword

The halfword operands are contained within the double word read into the J register from storage

(except for store halfword). The location of the halfword within the double word is specified by H register bits 21 and 22, which contain the storage address. The halfwords can be located as follows:



When H21 is 0 and H22 is 1, the halfword is located in the right half of the left word; when H21 is 1 and H22 is 0, the halfword is located in the left half of the right word, etc. H bit 23 must be off or a specification interrupt will occur.

In executing the halfword instructions, the operands are first expanded to a full word by propagating (extending to the left) the sign bit 16 positions. The expansion takes two cycles, defined by the first FXP trigger and Hwd Log trigger, and does not alter the value of the operand. After these two cycles, the execution of halfword instructions is identical to their full word counterparts.

During first FXP, the operand is gated through the shifter, expanded eight bits, and sent to the K register. During Hwd Log, the operand is gated through RBL, expanded the next eight bits, and sent back to the J register. The gating during the first FXP depends on H bits 21 and 22, which locate the operand in the J register. Figure 6305 shows the data flow during the first FXP and Hwd Log cycles, and detailed circuits.

LOAD HALFWORD (LH)

- The halfword operand 2 is expanded to a full word and placed in the operand 1 location.

The load halfword instruction is executed in four cycles, the first two being used to expand the halfword operand to a full word. The second two cycles transfer the operand to the K register and make the put-away to the general register specified by R1. At the start of execution, the second operand is located as follows:

Address (H 21 and 22)	Location
0 0	J0-15
0 1	J16-31
1 0	J32-47
1 1	J48-63

Figure Reference: Figure 6305.

First FXP

The left half of the TC side of the adder receives the output of J0-31 or J32-63, depending on H bit 21. Then the operand is shifted left 8 or right 8, depending on H bit 22, and the sign bit is propagated 8 positions left. The result, sent to the K register, has nine sign bits in its high-order followed by 15 integer bits.

Hwd Log

The partially expanded operand is gated from the K register to RBL. The RBL is then gated R8 to the J register with the sign propagated through the high-order byte. At the completion of this cycle, the fully expanded second operand is in the left half of the J register.

Hwd Add

The operand in the J register is gated through the adder to the K register.

ELC

A put-away is made to the general register specified by ER1 and the MODAR trigger is set.

ADD HALFWORD (AH)

- The halfword operand 2 is added to operand 1 and the algebraic sum is placed in the operand 1 location.
- CC is set to record the relation of the result to 0, and to record overflow.
- On an overflow, if the fixed-point overflow mask bit (PSW 36) is one, an interrupt is taken.

The add halfword instruction is executed in four cycles. The first two cycles are used to expand operand 2 to a full word. The second two cycles add the two operands and make the put-away to the general register specified by R1. At the start of execution, the operands are located as follows:

Operand 2 Address (H 21 and 22)	Operand 2	Operand 1
0 0	J0-15	M0-31
0 1	J16-31	M0-31
1 0	J32-47	M0-31
1 1	J48-63	M0-31

Figure Reference: Figure 6305.

First FXP

This cycle is identical to the first FXP cycle described for the halfword load instruction.

Hwd Log

This cycle is identical to the Hwd Log cycle described for the halfword load instruction.

Hwd Add

The second operand in the J register is added to the R1 operand in the M register. The result is sent to the K register.

ELC

A put-away is made from K register to the general register specified by ER1. MODAR is set, and a 0, 1, 2, or 3 is set into the condition register.

The condition register setting is determined in the same manner as for the full word add-subtract instructions; an examination is made of the K0 bit (sign bit), the K0-63 zero latch, and the carries from adder positions 1 and 0.

SUBTRACT HALFWORD (SH)

- The halfword operand 2 is subtracted from operand 1 and the algebraic difference is placed in the operand 1 location.
- CC is set to record the relation of the result to 0, and to record overflow.
- On an overflow, if the fixed-point overflow mask bit (PSW 36) is one, an interrupt is taken.

The subtract halfword instruction is executed in four cycles. The first two cycles are used to expand the second operand to a full word. The second two cycles subtract operand 2 from operand 1 and place the difference in the general register specified by R1. At the start of execution, the operands are located as follows:

Operand 2 Address (H 21 and 22)	Operand 2	Operand 1
0 0	J0-15	M0-31
0 1	J16-31	M0-31
1 0	J32-47	M0-31
1 1	J48-63	M0-31

Figure Reference: Figure 6305.

First FXP

This cycle is identical to the first FXP cycle described for the halfword load instruction.

Hwd Log

This cycle is identical to the Hwd Log cycle described for the halfword load instruction.

Hwd Add

Operand 2 in the J register and operand 1 in the M register are gated through the adder. The AM complement and hot 1 gates are activated to complement operand 2. The adder output (algebraic difference) is sent to the K register.

ELC

A put-away is made from the K register to the general register specified by ER1. The MODAR trigger is set, and a 0, 1, 2, or 3 is gated to the condition register.

The setting of the condition register is determined in the same manner as for full word add-subtract instructions; an examination is made of the K0 bit (sign bit), the K0-63 zero latch, and the carries from adder positions 0 and 1.

COMPARE HALFWORD (CH)

- The relation between the halfwords operand 1 and operand 2 determines the setting of the CC.
- No put-away is made.

The compare halfword instruction is executed in four cycles. The first two cycles are used to expand the second operand halfword to a full word. The second two cycles combine the two operands in the adder and set the condition code accordingly. At the start of execution, the operands are located as follows:

Operand 2 Address (H 21 and H22)	Operand 2	Operand 1
0 0	J0-15	M0-31
0 1	J16-31	M0-31
1 0	J32-47	M0-31
1 1	J48-63	M0-31

Figure Reference: Figure 6305.

First FXP

This cycle is identical to the first FXP cycle described for the halfword load instruction.

Hwd Log

This cycle is identical to the Hwd Log cycle described for the halfword load instruction.

Hwd Add

Operand 2 in the J register and operand 1 in the M register are gated through the adder. The AM complement and hot 1 gates are activated to complement operand 2. The result is sent to the K register for a zero check.

ELC

A put-away to a general register is prevented by turning on the block PA trigger. The MODAR trigger is set, and a 0, 1, or 2 is gated to the condition register.

The condition register inputs are determined in the same way as for the full word compare instructions. That is, an examination is made of the operand signs, the carry from the adder output 0 position, and of the state of the K0-63 zero latch. The sign indications are obtained from the R1 and R2 sign triggers (see Figure 5400). The interpretation given the above indications for setting the condition register is explained in the description of the full word compare instructions, under "Compare Logic."

STORE HALFWORD (STH)

- The halfword operand 1 is stored in the operand 2 location.
- The CC is not set.

If there is no delay in communicating with storage, the store halfword instruction is executed in two or three cycles, depending on the halfword storage address. If the operand is to occupy the left half of a full word, an additional cycle is necessary to position it in the K register. Because of this extra cycle, the store request is made during the first FXP (the first execution cycle) by the E unit, for this case (H22=0). For the two cycle execution (H22=1), the store request is made during T2, same as the full word store instruction. The reason for the delay in making the store request in the three-cycle case is that without the delay, storage would be ready to receive the K register output before the last execution cycle.

At the beginning of the execution, the R1 operand is located in M16-31.

Figure Reference: Figure 6306.

First FXP

The R1 operand in the M register is gated through the adder to the K register. Depending on the halfword address (H21 and H22), the operand is gated straight, left 8, right 8, or right 32 as it passes through the shifter.

At the completion of this cycle, the operand is either properly positioned (H22=1), or an additional eight-bit shift is required (H22=0). In the latter case, a store request is made by the E unit (E store request - KX 351). In the former case -- minimum two cycle execution -- the I unit made the store request during I to E transfer and the accept pulse can arrive during this cycle; if it does, the next cycle will be the ELC-store cycle.

To prevent a put-away, the block PA trigger is turned on at A pulse of the next cycle and stays on through the ELC cycle.

Hwd Log

This cycle is taken only for the minimum three-cycle operation (H22=1) for the final positioning of the halfword operand. The partially positioned operand in the K register is gated through the adder and shifted left 8 or right 8, depending on H bit 21. The result is gated back to the K register, ready for the transfer to the SBI latches.

Store

The store trigger is turned on after the first FXP trigger or Hwd Log trigger to define idle cycles while the E unit waits for an accept from BCU. The store trigger stays on for as long as the delay exists, and through the ELC cycle. If there is no delay, the store trigger will be on during ELC only.

The MODAR trigger is set during the store cycle(s).

ELC

The ELC trigger is turned on by the A clock following the accept pulse (which can arrive as early as first FXP for the two-cycle execution, or as early as Hwd Log for the three-cycle execution). At EB

time of ELC, the K register is set into the SBI latches.

The halfword that is selected for entry to storage depends on the mark register, which is set at I to E transfer on the basis of H bits 21 and 22.

A general register put-away is prevented by the block PA trigger, which is turned on at A pulse following the first FXP cycle.

T2 is allowed to be on during this cycle because the accept pulse had removed the block in the previous cycle (E TF block T2M on accept -- KX 515).

AND (N, NR)

- Corresponding bits in operand 1 and operand 2 are AND'ed to form a result that is placed in the operand 1 location.

- CC is set to record whether the result is zero or not zero.

A bit-for-bit comparison of the two operands is made to determine the result. When position 3 of both operands contains a 1 bit, for example, a 1 bit is placed in result position 3; when only one or neither of the operands contains a bit in position 3, the result bit for that position is 0.

The AND instruction is executed in three cycles, defined by the first FXP trigger, Hwd Log trigger, and ELC trigger. At the start of execution, the operands are located as follows:

Instruction Format	Operand 1	Operand 2
RR	M0-31	M32-63
RX-even	M0-31	J0-31
RX-odd	M0-31	J32-63

Figure Reference: Figure 6307.

First FXP

Operand 1 and operand 2 are gated to opposite sides of the adder and the logical AND function is performed at the input to the shifter. Because the shifter is used as the data path, a shift must be selected; the R4 shift is arbitrarily chosen and will be compensated for in the next cycle.

The result of AND'ing the two operands is gated to the K register and will occupy positions 4-35.

Hwd Log

This cycle is taken to compensate for the shift (R4) that was necessary during the first FXP, and to provide correct parity for the result. Incorrect parity probably resulted from the AND'ing operation because the halfsum outputs, which are used to generate parity, do not match the shifter input as they do when only one operand is sent through the shifter.

The result in the K register is sent to the adder, shifted left 4, and gated back to the K register to occupy positions 0-31. The sel log exc OR gate is active to allow the result through the shifter input without alteration. Because bad parity is expected, the check on the result as it is gated out of the K register, and the halfsum parity check are both blocked (KX 608).

ELC

A put-away is made from the K register to the general register specified by ER1. The MODAR trigger is set, and a 0 or 1 is gated to the condition register.

If the K0-63 zero latch is on, CR 35 is set (KX 661); if the latch is off, CR 35 is not set.

OR (O, OR)

- Corresponding bits in operand 1 and operand 2 are OR'ed to form a result that is placed in the operand 1 location.

- CC is set to record whether the result is zero or not zero.

A bit-for-bit comparison of the two operands is made to determine the result. For example, when position 3 of either or both operands contains a 1 bit, a 1 bit is placed in result position 3. When neither operand contains a bit in position 3, the result bit for that position is 0.

The OR instruction is executed in three cycles, defined by the first FXP trigger, Hwd Log trigger, and the ELC trigger. At the start of execution, the operands are located as follows:

Instruction Format	Operand 1	Operand 2
RR	M0-31	M32-63
RX-even	M0-31	J0-31
RX-odd	M0-31	J32-63

Figure Reference: Figure 6307.

First FXP - Hwd Log - ELC

These three cycles are the same as those described for the AND instruction. In addition to selecting the sel log AND gate in the first FXP, however, the sel log exc OR gate is also selected to perform the OR function at the shifter input. The R4 shift sets the result in K4-35.

During Hwd Log, the result in the K register is looped back through the adder to make a left 4 shift and generate proper parity. The K register output check and the adder HS parity checks are suppressed during this cycle. During ELC, the put-away is made, the MODAR trigger is set, and a 0 or 1 is gated to the condition register.

EXCLUSIVE OR (X, XR)

- Corresponding bits in operand 1 and operand 2 are exclusive OR'ed to form a result that is placed in the operand 1 location.

- CC is set to record whether the result is zero or not zero.

A bit-for-bit comparison of the two operands is made to determine the result. For example, when position 3 of either, but not both, operand contains a 1 bit, a 1 bit is placed in result position 3. When neither or both operands contain a 1 bit in position 3, the result bit for that position is 0.

The exclusive OR instruction is executed in three cycles, defined by the first FXP trigger, the Hwd Log trigger, and the ELC trigger. At the start of execution, the operands are located as follows:

Instruction Format	Operand 1	Operand 2
RR	M0-31	M32-63
RX-even	M0-31	J0-31
RX-odd	M0-31	J32-63

Figure Reference: Figure 6307.

First FXP

Operand 1 and operand 2 are gated to opposite sides of the adder and the exclusive OR function is performed at the input to the shifter with the sel log exc OR gate. Because the shifter is used as the data path, a shift must be selected; the R4 shift is arbitrarily chosen and will be compensated for in the next cycle.

The result of exclusive OR'ing the two operands is gated to the K register and will occupy positions 4-35.

Unlike the AND and OR instructions, proper parity is generated when the two operands are exclusive OR'ed. This is because the adder halfsums are identical, bit for bit, with the shifter input from the exclusive OR function. The halfsums are used in shifter operations to generate proper parity. Since this is a shifter operation, and the halfsums are the same as the shifter input for the exclusive OR function, proper parity is generated.

Hwd Log

This cycle is taken to compensate for the shift (R4) that was necessary during the first FXP. The result in the K register is sent to the adder, shifted left 4, and gated back to the K register to occupy positions 0-31. The sel log exc OR gate is active to allow the result through the shifter input without alteration.

The parity check on the result as it is gated out of the K register, and the halfsum parity check, are not blocked as in the AND and OR instructions. The result from the first FXP should contain proper parity, as explained above.

ELC

A put-away is made from the K register to the general register specified by ER1. The MODAR trigger is set, and a 0 or 1 is gated to the condition register.

If the K0-63 zero latch is on, CR 35 is set (KX 661); if the latch is off, CR 35 is not set.

SHIFT RIGHT SINGLE (SRA)

- R1 is shifted right.

- CC is set to record the relation of the result to 0.

The 31 low-order bits in general register R1 are shifted right the number of times specified by the six low-order bits of the effective address (B2 + D2). The vacated positions are filled with the sign bit, and bits shifted out of the register are lost. The details of this instruction are covered under "Logical Shift Left Double."

SHIFT RIGHT DOUBLE (SRDA)

- R1 and R1 +1 are shifted right as a pair of coupled registers.

- CC is set to record the relation of the result to 0.

The 63 low-order bits in the double-length operand contained in general registers R1 and R1 +1 are shifted right the number of places specified by the six low-order bits of the effective address (B2 + D2). The vacated positions are filled with the sign bit, and any bits shifted out of the low order are lost.

The R1 field of the instruction specifies an even/odd pair of general registers and must contain an even register address. If R1 is odd, the instruction is not executed, and a specification interrupt is taken instead.

The details of this instruction are covered under "Logical Shift Left Double."

SHIFT LEFT SINGLE (SLA)

- R1 is shifted left.
- CC is set to record the relation of the result to 0, and to record overflow.

The 31 low-order bits in general register R1 are shifted left the number of places specified by the six low-order bits of the effective address (B2 + D2). The sign remains unchanged, and any bits shifted out of position 1 that are unlike the sign bit cause an overflow.

The details of this instruction are covered under "Logical Shift Left Double."

SHIFT LEFT DOUBLE (SLDA)

- R1 and R1 +1 are shifted left as a pair of coupled registers.
- CC is set to record the relation of the result to 0, and to record overflow.

The 63 low-order bits in the double-length operand contained in general registers R1 and R1 +1 are shifted left the number of places specified by the six low-order bits of the effective address (B2 + D2). The sign remains unchanged, and any bits shifted out of position 1 of the left half of the double operand that are unlike the sign cause an overflow.

The R1 field of the instruction specifies an even/odd pair of general registers and must contain an even address. If R1 is odd, the instruction is not executed, and a specification interrupt is taken.

The details of this instruction are covered under "Logical Shift Left Double."

LOGICAL SHIFT RIGHT SINGLE (SRL)

- The logical operand in R1 is shifted right.
- The CC is not set.

The 32 bits in general register R1 are shifted right the number of places specified by the six low-order bits of the effective address (B2 + D2). Bits shifted out of the low order are lost, and zeros are supplied to the vacated high-order positions.

The details of this instruction are covered under "Logical Shift Left Double."

LOGICAL SHIFT RIGHT DOUBLE (SRDL)

- The logical operands in R1 and R1 +1 are coupled and shifted right.

The 64 bits in the double-length operand specified by general registers R1 and R1 +1 are shifted right the number of places specified by the six low-order bits of the effective address (B2 + D2). Bits shifted out of the low order of the double operand are lost and zeros are supplied to the vacated high-order positions.

The R1 field of the instruction specifies an even/odd pair of general registers and must contain an even address. If R1 is odd, the instruction is not executed, and a specification interrupt is taken.

The details of this instruction are covered under "Logical Shift Left Double."

LOGICAL SHIFT LEFT SINGLE (SLL)

- The logical operand in R1 is shifted left.
- The CC is not set.

The 32 bits in general register R1 are shifted left the number of places specified by the six low-order digits of the effective address (B2 + D2). High-order bits shifted out are lost and zeros are supplied to the vacated low-order positions.

The details of this instruction are covered under "Logical Shift Left Double."

LOGICAL SHIFT LEFT DOUBLE (SLDL)

- The logical operands in R1 and R1 +1 are coupled and shifted left.
- The CC is not set.

The 64 bits in the double-length operand specified by general registers R1 and R1 +1 are shifted left the number of places specified by the six low-order digits of the effective address (B2 + D2). High-order bits shifted out are lost, and zeros are supplied to the vacated low-order positions.

The R1 field of the instruction specifies an even/odd pair of general registers and must contain an even address. If R1 is odd, the instruction is not executed, and a specification interrupt is taken.

Circuit Description for All Shift Instructions

The following text is pertinent to all eight shift instructions: the single and double shifts, the logical and algebraic.

The number of cycles necessary to execute the shift instructions is dependent on the number of shifts to be taken, which can be as few as zero and as many as 63 shifts. The first execution cycle shifts the operand(s) from 1 to 8 shifts, leaving the number of remaining shifts to be taken a multiple of 8. The remaining shifts, therefore, are taken in increments of 8 per cycle. When shifting is completed, a single operand put-away is made for the single shift instructions, and a double operand put-away is made for the double shift instructions. The algebraic shift instructions also set the condition code.

Execution of the shift instructions is controlled by the first FXP, PA, and ELC triggers. At the start of execution, the R1 operand is located as follows:

Instruction Format	R1 Operand
Single	M0-31
Double	M0-63

Figure References: Figure 6308.

The first execution cycle is defined by the first FXP trigger and is known as the first cycle shift. In it, a number of shifts is taken so that the number of remaining shifts is a multiple of 8. The shifts taken during this first cycle is determined by examining all 6 bits of the effective address (B2 + D2), which specify the total shift:

Value:	32	16	08	04	02	01
	X	X	X	X	X	X

If there are one or more bits in the three low-order positions, a shift equal to their combined value is taken. If there are no bits in any of the three low-order positions, and there is at least 1 bit in the three high-order positions, a shift of 8 is taken. When the entire field is equal to 0, no shifts are taken.

The first cycle shifting is accomplished with the M register out-gates (St, R1, R2, and R3), referred to as bit shift gates, and with the main adder shift gates; the M register contains the operand(s). The shifts from these sources are selected so that the sum of the two will equal the required shift. For example:

Required Shift	M Out-Gate	Shifter Gate
R5	R1	R4
L5	R3	L8
R8	St	R8

The selection of these gates for all first cycle shifts is shown in a table in Figure 6308.

After the first cycle, the remaining shifts, if any, are taken in shifts of 8 per cycle. These cycles are known as shift iteration cycles and, like the first shift cycle, are defined by the first FXP trigger.

In the algebraic shift instructions, minus signs are propagated whenever the R1, R2, or R3 bit shift gate is used, and whenever the R4 and R8 shifter gate is used. Thus, in the algebraic right shift instructions, the proper sign occupies the vacated high-order positions of the result. In the algebraic left shift instructions, the sign is protected by blocking, in the shifter, the entry of bits into position 0. A shift out of position 1 of a bit unlike the sign bit turns on the shifter overflow trigger which results in a fixed-point overflow interrupt if the PSW bit 36 is 1.

Completion of the total required shifts is signaled by the shift counter (SC), which is loaded at the start of execution with the three high-order bits (whose value is a multiple of 8) of the 6-bit shift field (in the H register). This value is then decremented, by sending it through the exponent adder, by an effective value of 8 each time that an 8-shift is taken. Thus, the shift counter is decremented every shift iteration cycle, and possibly during the first cycle shift. Shifting is completed when the shift counter is equal to zero.

The signal used to end shifting and start the put-away sequencer, however, is called SC equal + less two, and is available in the cycle in which the last shift is being taken. The reason this signal is used is that the shift counter is decremented by 2, not 8, because of the positions (SC 4, 5, and 6) occupied by the three bits placed in it. Therefore, the three bits (in the shift counter) have a value whose multiple is 2 even though in the H register the same three bits have an 8 multiple value.

After the last shift cycle, one more cycle is taken in the single operand shift instructions to make the put-away and set the condition code for the algebraic type. Two more cycles are taken in the double operand shift instructions to put-away the operands, one in each cycle. The algebraic type also sets the condition code before termination.

First FXP -- First Cycle Shift

As previously explained, 8 shifts can be taken in this cycle. The operand(s) in the M register is gated through the proper out-gate to the adder, and through the shifter if necessary. The result is gated to both the K and M registers.

The choice of shift gates to be used is made by decoders on the basis of their input, which is the 6-bit shift amount. The shifter gate (R4, L4, R8, or L8), if any, is chosen by the shift counter decoder, which has the 6 bits available to it all through the

first cycle. The bit shift gate (St, R1, R2, or R3) is chosen by a separate decoder whose input, the two low-order bits of the six, is available during the T2 cycle to set the proper out-gate trigger at A time of this cycle. The shift gates chosen by the two decoders during the first cycle for all shift amounts are shown in a table in Figure 6308.

When the R1, R2, or R3 M register out-gate is chosen by the algebraic shift instructions, and the sign of the operand(s) is minus, 1 bits are forced through the M register outputs 0, 1, and 2, depending on the out gate. If the R1 gate is used, a 1 is forced out of position 0; if the R2 gate is used, a 1 is forced out of positions 0 and 1; if the R3 gate is used, a 1 bit is forced out of positions 0, 1, and 2. This sign propagation is accomplished by the M prop sign trigger (RM 001) that is turned on at the beginning of this cycle by the R1 sign trigger (Figure 5400)

When the R4 or R8 shifter gate is chosen by the algebraic right shift instructions, the sign is propagated in the shifter by the prop sign signal (KU 163). Propagation of the sign bit is similar to propagation of the 16 bit, which is illustrated in Figure 6305.

When the L4 or L8 shifter gate is chosen by the abgebraic left shift instructions, the sign is preserved in the shifter with the save sign signal (KU 163). That is, the M register bit 0 is sent through to AM output latch 0 regardless of the shift; also, a shift into position 0 from positions 4 or 8 is blocked (AQ 101). In addition to these, the save sign signal will turn on the shifter overflow trigger if it is found that a bit unlike the sign bit was shifted out of position 1 (AQ 311). The instruction is not altered or shortened as a result of turning on the shifter overflow trigger.

The shift counter is gated to the exponent adder, decremented by 2 if a shift of 8 is made in this cycle, and returned to the shift counter. A shift of 8 is possible if the three low-order bits of the 6-bit shift field are zero and there is a bit in at least one of the three high-order bits. If this were the only shift required, the signal SC equal + less two would be active in this cycle, and would AND with the $\overline{H21 + H22 + H23}$ signal to raise the shift complete level (Figure 6308). If the total shift required is from 0 to 7, the SC equal + less two signal AND's with the SC eq zero lth to raise the shift complete level.

First FXP -- Shift Iteration

If, after the first cycle shift is completed, additional shifting is required, a shift iteration cycle(s) is taken. The R1 operand in the M register is gated to the main adder, shifted 8, and returned to the K and M registers. Sign propagation for the algebraic right shifts, and the save sign function for the algebraic left shifts operate in these cycles as described in "First FXP - First Cycle Shift."

The shift counter is gated to the exponent adder, decremented by 2, and returned to the shift counter. Iteration cycles are continued until the signal SC equal + less two is generated, indicating a value of 0 after completion of the current cycle. The signal AND's with first cycle memorized latch (Figure 6308) to raise the shift complete signal that starts the put-away sequence(s).

Decrementing the shift counter is accomplished by adding its complement value to 2, and then re-complementing to obtain a true result. Both complement (1's) operations are accomplished in one pass through the exponent adder. The shift counter is gated to the TC side and a 2 (6 bit) is inserted in the AM side. The following example shows a shift counter value 10 decremented to 8:

SC Positions	4	5	6	7	
Values	8	4	2	1	Value
SC Content	1	0	1	0	(10)
Comp (1's) SC at Input	0	1	0	1	(5)
Add Six Bit	0	0	1	0	(2)
Sum	0	1	1	1	(7)
Comp (1's) Sum	1	0	0	0	(8)

The second complement gate complements the input (sum) to the exponent adder output latches. The reduced shift counter value is effective in the next cycle, as the shift counter is released at A clock.

PA -- First Put-Away

This cycle follows the last shift cycle for the double-operand shift instructions. In this cycle, the first put-away is made, preparations are completed for the second put-away, and the MODAR trigger is set. In addition, for the algebraic instructions, the condition code input is gated, and the fixed-point overflow latch can be set.

The first operand put-away is made from the K register, which had been receiving the result along with the M register during the shift cycles, to the general register specified by ER1. In preparation for the second put-away, the M register right half is gated through its left 32 out gate, to the adder and sent to the K register left half; and the ER1 register is incremented to contain the R1 +1 value (KX 535).

For the algebraic instructions (SRDA and SLDA), the results of the tests on whether the double operand is equal to, less than, or greater than zero, are gated to the condition register. The K0 bit and the K0-63 zero lth are tested (KX 635 and KX 655). In

addition, if the shifter overflow trigger (KS 141) was set during the shift cycles in the SLDA instruction, the condition code will be set to 3, and, if the PSW bit 36 is 1, the fixed-point overflow latch is set to initiate an interrupt sequence.

ELC -- Second Put-Away

This is the second put-away cycle for the double-operand shift instructions. The R1 +1 operand (sent to the K register left half in the previous (PA) cycle) is set into the general register specified by ER1, which now contains the R1 +1 address.

PA and ELC -- Put-Away for Single Operand Instructions

The operand in the K register left half is set into the general register specified by ER1. The MODAR trigger is set, and, for the algebraic instructions (SRA and SLA), the setting of the condition register is determined according to the operand's relation to zero (KX 635 and KX 655). In addition, if the shifter overflow trigger (KS 141) was set during the shift cycles in the SLA instruction, the condition code will be set to 3, and, if PSW bit 36 is 1, the fixed-point overflow latch is set to start an interrupt sequence.

MULTIPLY (M, MR)

- Operand 1 is multiplied by operand 2; the double word product is placed in R1 and R1 + 1.
- CC is unchanged.

A 32-bit multiplier (second operand) and a 32-bit multiplicand (first operand) form a 64-bit product that is placed in the general registers R1 and R1 + 1. The R1 field of these instructions must therefore specify an even register, or a specification interrupt will occur. The multiplicand is taken from the R1 + 1 register. The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs; however, a zero result is always positive.

The Introduction for this instruction is included in "Multiply Halfword," below. Details are located in Figure 6309.

MULTIPLY HALFWORD (MH)

- The halfword operand 1 is multiplied by the halfword operand 2; the full word product is placed in R1.
- CC is unchanged.

A 16-bit multiplier (second operand) and a 16-bit multiplicand (first operand) form a 32-bit product that replaces the multiplicand in R1. The multiplier is expanded to a full word before multiplication by propagating its sign bit.

If the multiplicand is greater than 16 bits (if any of its high-order bits are unlike the sign bit), the product will be greater than 32 bits. Because only the low-order 32 bits are stored in R1, the product could be invalid; however, no overflow indication is given for the lost bits.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs; however, a zero result is always positive.

Introduction

The following text is pertinent to all three multiply instructions (MR, M, and MH).

Multiplication is performed by multiplying the multiplicand by each hexadecimal group in the multiplier. The result of these multiplications, called partial products, are added together as they are formed until the last hexadecimal group adds its partial product to the rest to form the final product. Two housekeeping cycles precede the multiplications, known as iterations (repetitions), for the MR and M instructions. For the MH instructions, four housekeeping cycles precede the iterations, the first two taken to expand the halfword multiplier (see "Halfword Expansion"), with the second two cycles being identical to the two housekeeping cycles for the MR and M instructions. The housekeeping cycles are followed by eight iterations, one for each multiplier group; the iterations take one or two cycles, depending on the multiplier group. Following the iterations, two more cycles are taken for the put-aways.

If either the multiplier or multiplicand is 0, the condition is detected in the last housekeeping cycle, and no iteration cycles are taken. Instead, a zero product is stored in the specified general register(s).

At the start of execution, the operands are located as follows:

Instruction		Operand 2 (Multiplier)	Operand 1 (Multiplicand)
(MR)*	RR	J32-63 (R2)	M32-63 (R1 + 1)
(M)	RX-even	J0-31	M32-63 (R1 + 1)
(M)	RX-odd	J32-63	M32-63 (R1 + 1)
(MH)	RX-H21 and 22		
	00	J0-15	M0-31 (R1)
	01	J16-31	M0-31 (R1)
	10	J32-47	M0-31 (R1)
	11	J48-63	M0-31 (R1)
After two-cycle expansion:		J0-31	

*This is a GROUT class instruction. The multiplicand is delivered during T2, and the multiplier is delivered during first FXP. Both operands cannot be delivered in one cycle because they are both sent over GBR.

The block T2M trigger is turned on and remains set for most of the execution to protect the J register, which contains the multiplier, from an operand fetch by the I unit. In single cycle, the block T2M trigger remains on all during the execution.

Figure Reference: Figure 6309.

Multiplication is performed by accumulating multiples of the multiplicand according to the value of each hexadecimal digit in the multiplier, beginning with the low-order digit. The application of this method using decimal numbers is:

```
Multiplicand:   462
  Multiplier:  x285
                2310   First PP and 5X Multiplicand
               3696                8X Multiplicand
               39270  Second PP
              924                  2X Multiplicand
              131670 Final Product
PP: Partial Product
```

This method is a slight variation of the familiar method of longhand multiplication. The difference is that intermediate totals are produced (partial products) instead of only one total.

The multiplicand multiples used in this example are referred to as the 5X, 8X, and 2X multiples. However, because these numbers are offset to the left when added to the partial products, their value is actually 5, 80, and 200 times the value of the multiplicand, respectively. Therefore, each multiplier digit to the left of the decimal point is increased by the power of 10 beginning with 10^0, then 10^1, 10^2, etc. The same is true of hexadecimal digits; each multiplier digit to the left of the decimal point is increased by the power of 16 beginning with 16^0, then 16^1, 16^2, etc. In the CPU, raising the multiplicand multiples by these powers is achieved by shifting right the partial products one hexadecimal digit (4 bits) as they are added to the multiples. The following example illustrates the same problem (462 x 285) using binary numbers divided into hexadecimal groups:

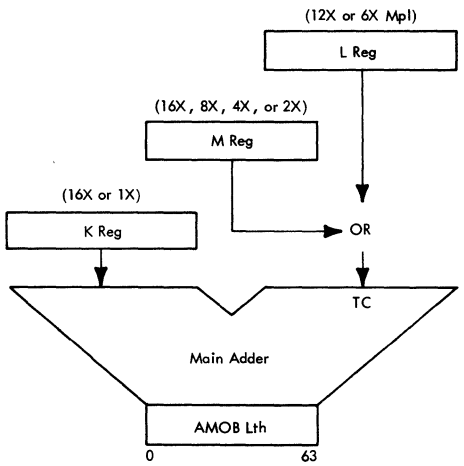
```
Multiplicand:  0001  1100  1110
                (1)   (1)   (13)
  Multiplier:  0001  0001  1101
       0001    0111  0111  0110  First PP and 13X Multiple
       0001    1100  1110              1X Multiple
0000   0011    0100  0101  0110  Second PP
0001   1100    1110                    1X Multiple
0010   0000    0010  0101  0110  Final Product
```

Because the partial products are right shifted as they are added, the value of the 13X and 1X multiples are 13, 16, and 256 times the value of the multiplicand, in the order in which they are used.

Arriving at a final product, therefore, amounts to adding together the partial products formed by the multiplicand and each of the hexadecimal groups in the multiplier.

If all 16 multiples of the multiplicand are available, one cycle (called an iteration cycle) would be taken for each of the eight hexadecimal multiplier digits to add the proper multiples to arrive at the final product. Because all multiples are not available, at least not after the first cycle, some compensations are made.

The multiples that are available throughout the execution are the even multiples because these are easily obtained. In the first execution cycle, the multiplicand is set into the high-order half of the K and M registers. If the multiplicand is now the 16X multiple of itself, the 16X, 8X, 4X, and 2X multiples are obtained from the M register by using the the St, R1, R2, and R3 out gates, respectively; and the 16X and 1X multiples are available from the K register by using the St and R4 gates. In the second execution cycle, the 12X multiple is derived by complement-adding the 4X multiple from the M register, to the 16X multiple from the K register, and placing the result in the L register. After this cycle, the L register can now supply the 12X multiple by using its straight gate, and the 6X multiple by using its R1 gate. Thus, individually, the K, L, and M registers can supply the 1X, 2X, 4X, 6X, 8X, 12X, and 16X multiples of the multiplicand as shown in the following illustration.



For the first iteration, these registers can individually or in combination supply any required multiple. The 5X multiple, for example, is obtained by combining the 1X multiple from the K register and the 4X multiple from the M register; the 10X multiple is obtained by using the 8X multiple, followed by the 2X multiple in the next cycle; the 14X multiple is obtained by using the 8X multiple, followed by the 6X multiple in the next cycle. For the 10X and 14X multiples, therefore, 2-cycle iterations take place. After the first iteration, the 1X multiple is no longer available because the K register is used to accumulate the partial products. Therefore,

the next seven iterations are accomplished with only the even multiples (2X, 4X, 8X, 10X, 12X, 14X, and 16X). Before showing how this is done, it would be worthwhile to understand why the multiplicand in the M register can be used as the 16X multiple of itself. If, as shown in the following example:

Iterations	K Register
1st	X X X X X X X X O O O O O O O O
2nd	O X X X X X X X X O O O O O O O
3rd	O O X X X X X X X X O O O O O O
4th	O O O X X X X X X X X O O O O O
5th	O O O O X X X X X X X X O O O O
6th	O O O O O X X X X X X X X O O O
7th	O O O O O O X X X X X X X X O O
8th	O O O O O O O X X X X X X X X O

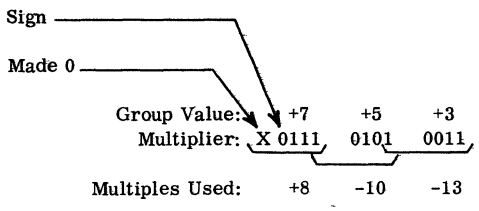
The O's and X's represent four-bit positions, or one hexadecimal digit. If either, but not both operands are negative, all digits to the left of X's would be F's.

the multiplicand is sent to the K register, via the adder, using the M St out-gate, it would occupy positions 0-31 of the K register. If this move represented the first of eight iterations, and if thereafter only the K register is gated to the adder, the additional seven iterations would shift the multiplicand in the K register an additional 28 bits, until it occupied positions 28-59 (the shifts are accomplished when the K register is gated R4 to the adder and returned to itself). The multiplicand is now 16 times its former value because of its final position in the K register. Therefore, if the R1, R2, or R3 out gates were used in the first transfer, the final value would have been 8, 4, or 2 times the multiplicand just as if the total multiplier value were 8, 4, or 2.

With only even multiples available, odd multiplier hexadecimal groups, except the first, are decoded to the next higher value. For example, a 3 decodes as a 4, a 5 decodes as a 6, etc. Therefore, if a hexadecimal group is odd an overmultiplication is made. To compensate for this, the preceding group is undermultiplied.

The decoding of each hexadecimal group, including the low-order group, examines the low-order bit of the next higher group to determine if it is odd. If it is, a 16 is deducted from the value of the group being decoded in anticipation of the overmultiplication by the same amount in the next multiplier group. An undermultiplication of 16 in one group, therefore, balances an overmultiplication of 1 in the next higher multiplier group.

No compensation is necessary for the low-order multiplier group, because, as already mentioned, all multiples of the multiplicand are available for the first iteration. The following example shows the value of the hexadecimal groups in a 12-bit multiplier versus the multiples used to participate in the iteration:



Total Multiplier Value (decimal)

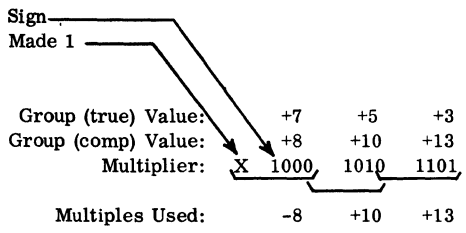
Calculated by Group Values	Calculated by Multiples Used
7 x 256 = 1792	+8 x 256 = +2048
5 x 16 = 80	-10 x 16 = -160
3 x 1 = 3	-13 x 1 = -13
+1875	+1875

The bit combination for the low-order hexadecimal group in the above example is equal to 3, but because the next higher group is odd, the multiple chosen for the first iteration is -13 (3-16 = -13). The next group is equal to 5 but decodes as a 6; because the next higher group is odd, the multiple chosen for the second iteration is -10 (6-16 = -10). The next hexadecimal group is equal to 7 but decodes as an 8; because this is the last group of a positive multiplier, the high-order bit is decoded as 0 and the multiple chosen for the third iteration is 8.

Negative multiples are achieved by raising the complement and hot 1 gates as the selected multiples are sent through the TC side of the adder. The high-order (5th) bit in the group that is being decoded alone determines the complement gating, as shown in the table in Figure 6309.

When either or both operands are negative, the product will be correct and with the proper sign without altering the multiplier decoding or changing any of the routing. Two negative or two positive operands produce a positive product; a positive and a negative operand produce a negative product. To understand why no compensation for negative operands is necessary, consider the following 12-bit negative multiplier, whose absolute decimal value

(-1875) is the same as the multiplier used in the preceding example (+1875):



Total Multiplier Value
(calculated in decimal)

-8 x 256 = -2048
+10 x 16 = +160
+13 x 1 = +13
-1875

Note that the multipliers in the two examples select the same multiples, but with opposite signs. This, of course, is what makes the multipliers equal in absolute value but different in sign. Therefore, the multiplier decoding as shown in the preceding table is valid for both positive (true) and negative (complement) multipliers.

Whether the final product is a positive or a negative number is taken care of by the high-order significant hexadecimal group in the multiplier. If the multiplier is positive, this group selects a positive multiple (+13, for example). Therefore, if the multiplicand is positive, the sign of the final product will be positive because this last added multiple of the multiplicand has a greater value than the partial product so far developed. If the multiplicand is negative, the final product will also be negative, again because the final multiple has a greater value than the partial product it is added to. When the multiplier is negative, the high-order hexadecimal group selects a negative multiple (-13, for example). This complements the selected multiplicand multiple so that if it was positive, it becomes negative, resulting in a negative final product; if it was negative, it becomes positive, resulting in a positive final product.

Additional iterations for high-order zeros in the multiplier extend the sign bit of the product. Only the K register, which contains the product, is gated to the adder. Negative signs are extended into adder inputs 0, 1, 2, and 3 because the R4 gate is used. Negative sign bits are also extended in the M and L registers in iterations in which they are gated to the adder via any of their right-gates.

The multiplier is decoded from the J register bits 27-31 or 59-63, depending on which half of the J register the multiplier is located. The J register is right-shifted four bits to supply a new 5-bit group for decoding before the iterations that use the multiples they select. The J register left shifts are accomplished by transferring the multiplier in J L4 to the RBL (-4 to 59) and returning to the J register with an R8 shift.

The iteration count is kept by the shift counter, which is initially set to a value of 8. The shift counter is reduced by 1 through the exponent adder, for each iteration taken until it reaches a value of 0. The first iteration is taken when the SC equals 8, and the last iteration is taken when the SC equals 1. The halfword multiply instruction also sets the shift counter to 8, even though the first 16 bits of the expanded halfword multiplier are equal to 0.

When either operand is zero, iteration cycles are not taken and the general register(s) receives a zero as a final product. If the multiplicand is 0, it is stored in the general registers R1 and R1 + 1 for the M and MR instructions, and in general register R1 for the MH instruction. If the multiplier is 0, the multiplicand is reset to zero before the put-away(s).

A detailed description of the multiply cycles is contained in Figure 6309 with the register flow charts.

DIVIDE (D, DR)

- The double word operand 1 (dividend) is divided by the single word operand 2 (divisor). The result is a one-word quotient and one-word remainder.
- CC is unchanged.

The dividend (first operand) is divided by the divisor (second operand) and replaced by the quotient and remainder.

The dividend is a 64-bit signed integer and occupies the even/odd pair of registers specified by the R1 field of the instruction. A specification exception occurs when R1 is odd. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered registers, respectively. The divisor is a 32-bit signed integer.

The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. All operands and results are treated as signed integers.

Introduction

To divide one number (dividend) by another (divisor) the dividend is repeatedly reduced by subtracting the divisor. The number of times this can be done is the solution (quotient), and anything left of the reduced dividend (some number greater than zero but less than the divisor) is called the remainder.

There are two basic methods of binary division. The two methods, restore and non-restore, are illustrated below.

Restore

Quotient: 0 0 1 1 0

Divisor 0 1 1 1, Dividend 0 0 1 0 1 1 0 1

Comp divisor		1 0 0 1	1st Iteration
Comp result (discarded)	n/c	1 0 1 1	
True partial dvd		0 1 0 1	
Comp divisor		1 0 0 1	2nd iteration
Comp result (discarded)	n/c	1 1 1 0	
True partial dvd		1 0 1 1	
Comp divisor		1 0 0 1	3rd iteration
True result	c ↖	0 1 0 0	
True partial dvd		1 0 0 0	
Comp divisor		1 0 0 1	4th iteration
True result	c ↖	0 0 0 1	
True partial dvd (remainder)		0 0 1 1	
Comp divisor		1 0 0 1	5th iteration
Comp result (discarded)	n/c	1 1 0 0	

Non-Restore

Quotient: 0 0 1 1 0

Divisor 0 1 1 1, Dividend 0 0 1 0 1 1 0 1

Comp divisor		1 0 0 1	1st iteration
Comp result	n/c	1 0 1 1	
Comp partial dvd		0 1 1 1	
True divisor		0 1 1 1	2nd iteration
Comp result	n/c	1 1 1 0	
Comp partial dvd		1 1 0 1	
True divisor		0 1 1 1	3rd iteration
True result	c ↖	0 1 0 0	
True Partial dvd		1 0 0 0	
Comp divisor		1 0 0 1	4th iteration
True result	c ↖	0 0 0 1	
True partial dvd (remainder)		0 0 1 1	
Comp divisor		1 0 0 1	5th iteration
Comp result	n/c	1 1 0 0	
Comp partial dvd (remainder-divisor)		1 1 0 0	
True divisor		0 1 1 1	correction cycle
True result (remainder)	c ↖	0 0 1 1	

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

In restore division, the result of any reduction of the dividend by the divisor is retained only if the result is the true difference (carry). This result, called the partial dividend, is used in the next reduction (iteration). However, if the result is the 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 bit is inserted in the quotient when the result is true (carry), and a zero bit is inserted when the result is complement (no carry).

In the non-restoring method of division, the result of an iteration is retained as the new partial dividend whether it is true or complement. When a partial dividend is true, the 2's complement divisor is added to it; when the partial dividend is complement, the true divisor is added to it. In each iteration, the partial dividend is shifted left one bit in relation to the divisor; also, a 1 bit is inserted in the quotient when the result is true (carry) and a 0 bit is inserted in the quotient when the result is complement (no carry).

From now on, use of the term dividend will mean both the initial and partial dividends.

Shifting the dividend left one bit doubles its value and is equivalent to halving the divisor. Similarly, shifting the dividend left two bits quadruples it and is equivalent to reducing the divisor to 1/4 of its value. Note the similarity between the restore and non-restore methods of division (the first successful reduction of the dividend is made by 1/4 of the divisor):

(Restore) Dividend -1/4 Divisor, is equivalent to
(Non-Restore) Dividend -Divisor + 1/2 Divisor + 1/4 Divisor

The Non-Restoring Method of Division, used by the 2075 CPU, is shortened in a way requiring half as many iterations. Every iteration, therefore, produces two quotient bits and shifts the dividend left two bit positions with respect to the divisor. The speed-up is done by reducing the dividend by multiples of the divisor (0X, 1/2X, 3/4X, 1X, 3/2X). The multiples are selected for each iteration by comparing the first three bits of a bit-normalized divisor (constant throughout the iterations) against the three comparable bits of the dividend (changing with each iteration).

One of the qualifications for the participating divisor multiple is that it reduce the dividend enough to make its two high-order bits insignificent (0's for a true result, and 1's for a complement result). This allows the dividend to shift left two bits for the next iteration.

If the dividend contains a significant bit in the high-order position (1XX for a true dividend, 0XX for a complement dividend) before an iteration, the reduced dividend is less than 1/4 of its value after the iteration (00X true result, 11X complement result).

The Divisor Multiples that insure a reduction of the dividend to less than 1/4 of its previous value in each iteration are shown below. The divisor multiples are subtracted from a true dividend and added to a complement dividend.

True Dividend	Comp Dividend	Divisor 111	110	101	100
111	000	1	1	3/2	3/2
110	001	1	1	1	3/2
101	010	3/4	3/4	1	1
100	011	3/4	3/4	1	1
011	100	1/2	1/2	1/2	1/2
010	101	1/2	1/2	1/2	1/2
001	110	0	0	0	0
000	111	0	0	0	0

Multiple Selection Table

These divisor multiples fulfill a second requirement for a two-bit divide; that is: when they are used, at least two quotient bits can be predicted. When the 0X, 1/2X, 1X, or 3/2X divisor multiples are used, two quotient bits are predicted; when the 3/4X divisor multiple is used, three quotient bits can be predicted. This third quotient bit takes the place of the first quotient bit predicted by the divisor multiple used in the next iteration.

In the table below, the quotient bits produced in a given iteration depend on: the divisor multiple used, on whether the partial dividend is true or complement, and on whether the iteration result (new dividend) is true or complement.

Partial Dividend	Iteration Result	Multiple Used 3/2	1	3/4	1/2	0
T	T	11	10	011	01	00
T	C	10	01	010	00	-
C	T	01	10	101	11	-
C	C	00	01	100	10	11

Quotient Prediction Table

Using the Quotient Prediction and the Multiple Selection Tables: The example below shows how the IBM 2075 CPU handles the same divide problem shown earlier. Before doing any of the iterations, the machine complements the divisor and dividend if they are negative numbers (2's complement). The quotient developed, therefore, is a positive number and is complemented if the divisor and dividend signs are unlike. In addition, the divisor is hex-normalized (all high-order hex-zeros removed) and the same number of hex zeros are removed from the high-order end of the dividend. (The dividend must have at least as many high-order zeros as the divisor, or the instruction is terminated with a divide check.) In comparing the leading divisor and dividend bits to select the divisor multiples, the divide decoder effectively bit-normalizes the divisor by ignoring its high-order zero's (up to 3) and ignores the same number of bit positions in the dividend. The divisor multiples, excepting the 0X multiple, derive from the M and L registers with their straight and right 1 out-gates to the TC side of the adder.

The first and second termination cycles, which follow the last iteration cycle, calculate a true remainder if it did not result from the last iteration cycle. In addition, the first termination cycle decides the last quotient bit when the last iteration cycle does not use the 3/4X divisor multiple.

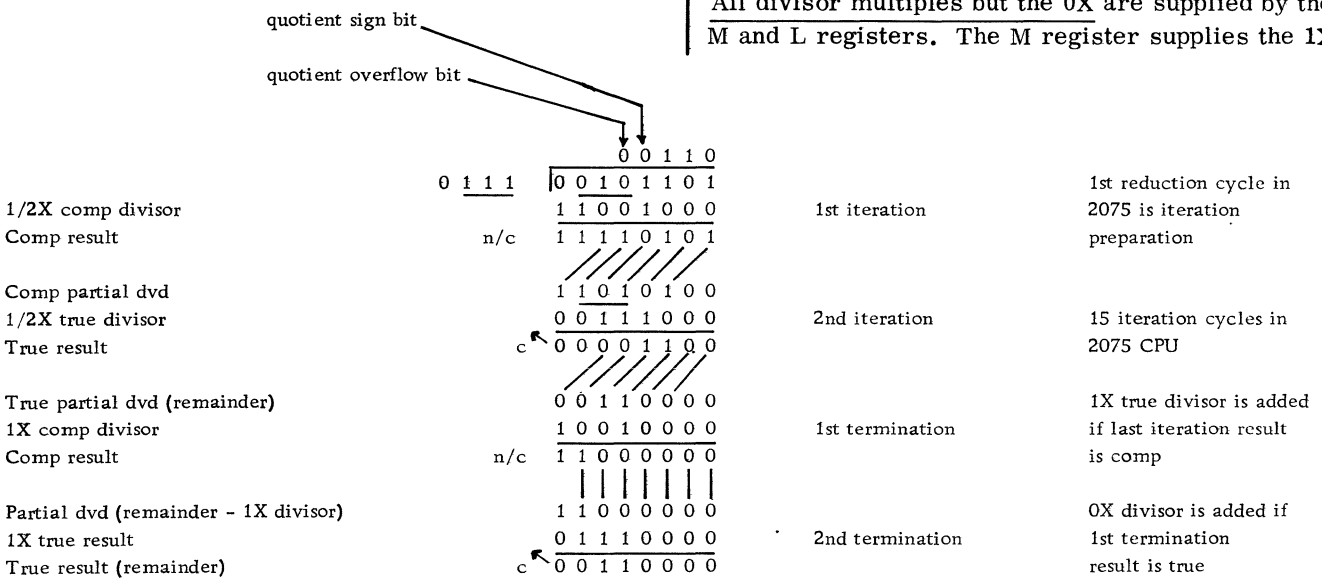
After the second termination cycle, and before the quotient and remainder are put away, the machine (1) complements the quotient if the divisor and dividend signs are unlike, (2) shifts the remainder right as much as the divisor and dividend were shifted left, to position the remainder correctly within the register, and (3) complements the remainder if the dividend is a negative number.

If the Dividend is Zero: No iterations take place and a zero quotient and remainder are placed in the R1 and R1 + 1 general registers.

Throughout the Divide Instruction: The machine checks for the possibility of a quotient larger than 32 bits. Detecting this condition terminates the instruction by forcing the PA cycle, followed by ELC, and causes a fixed-point divide interrupt. One of the checks made is inspecting the first quotient (overflow) bit; this bit must be 0, as it is outside a 32 bit quotient. The second quotient bit is the sign bit, and must also be zero unless the quotient is the maximum negative number (100 -- 00) and the divisor and dividend signs are unlike. The divide checks are further explained in the Divide Checks circuit diagram, Figure 6310.

All divisor multiples but the 0X are supplied by the M and L registers. The M register supplies the 1X

2-Bit Divide as Performed by the 2075 CPU (4-Bit Registers are Used)



Divisor Multiples

True	Complement (2's)	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0X:
0 1 1 1 0 0 0 0	1 0 0 1 0 0 0 0	1X:
0 0 1 1 1 0 0 0	1 1 0 0 1 0 0 0	1/2X:
1 0 1 0 1 0 0 0	0 1 0 1 1 0 0 0	3/2X:
0 1 0 1 0 1 0 0	1 0 1 0 1 1 0 0	3/4X:

Note: The first and second termination cycles produce a true remainder if it did not result in the last iteration cycle. Also, the first termination cycle produces the final quotient bit if the last iteration cycle did not use the 3/4X multiple.

multiple with its straight out-gate, and the 1/2X multiple with its right 1 out-gate. The L register supplies the 3/2X multiple with its straight out-gate, and the 3/4X multiple with its right 1 out-gate. The 0X multiple is merely a forced parity to the AMTC side of the adder. Selecting these multiples is the job of the divide decoder, which, during the iterations, compares the first three bits of a bit-normalized divisor against the comparable three bits of the partial dividend indicated in the 2-bit divide example.

Decoding the Divisor and Dividend Bits to select a new divisor multiple in each iteration is complicated by late availability of the new partial dividend (result) in the cycles. This new partial dividend must be inspected quickly in order to be compared with the divisor bits, and to select one of the M or L register out-gates (or force parity for the 0X multiple) to the adder by the next A pulse. To overcome delay, the decoder examines the adder sums 2 - 7 in each iteration when a possible carry to this group has not arrived. (The sums 0 and 1 need not be examined as these are both 0's or 1's and are discarded. All 6 bits of the adder result (2 - 7) must be presented to the decoder: The sums 2, 3, and 4 may be ignored by the decoder because the hex-normalized divisor can have up to three leading zeros.)

Using these bits, the decoder compares the three divisor bits against four possible sets of dividend bits:

1. No carry into the group $\overline{\text{(CG 4 - 7)}}$ and result true (carry out 0)
2. No carry into the group $\overline{\text{(CG 4 - 7)}}$ and result complement ($\overline{\text{carry out 0}}$)
3. Carry into the group (CG 4 - 7) and result true (carry out 0)
4. Carry into the group (CG 4 - 7) and result comp ($\overline{\text{carry out 0}}$)

Having considered these possibilities, the divide decoder is ready to select the next divisor multiple when CG 4 - 7 or CG 4 - 7 and the carry out 0 or carry out 0 signals become available near the end of the cycle.

The divide decoder is further explained in "Functional Units," 2075 Processing Unit -- Volume 1, FE Manual of Instruction, Form 223-2872.

The circuit description for the fixed-point divide instructions is in Figure 6310. It also contains the flow chart, register and circuit diagrams of the fixed-point divide instructions.

I EXECUTE

INTRODUCTION

The instructions covered in this section are executed by the I-execution unit alone, or jointly with the E unit. The instruction BCR with R2 equal 0 uses an IE unit sequencer but is covered in the Branch section.

The I-execution unit uses a set of six sequencers: IE1, IE2, IE3, IM1, IM2, and IEL. The functions performed by these sequencers depend on the instructions using them. Mainly, these sequencers are employed to use I-unit mechanisms. For example:

1. Gate PSW to the incrementer
2. Compute memory addresses
3. Gate out general registers
4. Increment BR1
5. Initiate store and fetch requests.

The names of most signals that control these and other functions are characterized by the prefix IE: IE fetch to J, IE gate H to incrementer, IE TF block ICM, etc.

THEORY OF OPERATION

LOAD PSW (LPSW)

The double word operand specified by B1 + D1 replaces the current PSW. The operand address must have its three low-order bits zero to designate a double word; otherwise, a specification interrupt will occur. In addition, since LPSW is a privileged instruction, the CPU must be in the supervisory state (current PSW bit 15 is 0) to execute it. If the CPU is in the problem state (current PSW bit 15 is 1), a privileged operation interrupt will occur instead

On completion of the LPSW instruction, and if the new PSW bit 14 is 0 (running state), an IC recovery sequence is initiated to fetch the instructions specified by the new ICR value (bits 40-63). Therefore, a valid storage address check for the new ICR value is made during the IC recovery sequence, and an even byte address check is made during the subsequent instruction times.

The CPU enters the problem state when the new PSW bit 15 is 1, and it similarly enters the wait state if the new PSW bit 14 is 1. Load PSW is the only instruction available for entering the problem or wait states.

Condition Code: The code is set according to bits 34 and 35 of the new PSW.

Program Interrupts: Privileged operation, addressing, and specification.

Details

The load PSW instruction is executed by the I unit and takes four execution cycles to complete. At T1, the address of the PSW is calculated, a fetch request is made, and the ID block IC line is activated to turn on the block ICM latch (System KA 141) at TN T2. IC fetches are blocked to prevent an unnecessary fetch because an IC recovery will probably be made at the end of the instruction execution; they are also blocked to prevent the use of the incrementer because the incrementer will be used to check the new PSW parity. T2 is turned on and the turn on of the block T1M trigger is gated. T1 is blocked to prevent the preparation of any new instruction, again because of the probability of an IC recovery at the end of the load PSW instruction. With the receipt of accept from BCU, the I go is generated by the I unit to turn on the first sequencer, IE1.

Figure Reference: Figure 6350.

IE1

The MODAR trigger is turned on, and, when J is loaded and valid, the J register is transferred to the PSW register and the channel interrupt priority circuits are reset.

The new PSW could change the system mask (PSW bits 1-7) that allows channel and external interrupt conditions to be recognized. Therefore, a channel that developed an interrupt condition and was granted channel priority during the execution of load PSW under control of the old system mask, must have that priority cancelled to allow the new system mask to control the channel interrupt requests.

In case of an invalid address error, or a SAP fetch error, the J register receives the output of the panel keys (with correct parity) instead of the new PSW. The J register is then termed invalid and effective execution of the load PSW instruction is blocked.

IE2

The J loaded trigger is turned off, and, if J is valid, the right half of the PSW register (32-63) is gated to the incrementer for a parity check, and the incrementer check trigger is enabled.

IE3

If J is valid, the left half of the PSW register (0-31) is gated to the incrementer and the incrementer check trigger is enabled. If the new PSW bit 14 is

0 (running state), the IC recovery trigger is gated on to start a recovery sequence. The IC recovery trigger turns off the block T1M trigger to prepare for the time when IOP is loaded with the first newly fetched instruction. The IC recovery trigger also turns off the block ICM trigger to allow the new ICR value to pass through the incrementer into SAR for the instruction fetch.

Any interrupt will take priority over the IC recovery sequence.

IEL

Gate the turn off of the MODAR trigger. If the new PSW bit 14 is 0, gate the turn on of the wait trigger; if the new PSW bit 14 is 1, turn on the IC recovery trigger at the beginning of this cycle.

SET PROGRAM MASK (SPM)

Bits 2-7 of the general register specified by the R1 field replace the condition code (34 and 35) and the program mask bits (36-39) of the current PSW.

Condition Code: The code is set according to bits 2 and 3 of the general register specified by R1.

Details

The set program mask instruction is executed by the I unit and takes three execution cycles to complete. At T2, T1 is blocked to prevent an address calculation from interfering with the readout of GR R1 during the execution of this instruction. The signal I go, generated by T2, turns on the GROUT and IE1 triggers. GROUT is used to gate out the specified general register(s) during execution cycles.

Figure Reference: Figure 6351

IE1

The GROUT trigger gates out GR R1 (and R2 which is not used), from which bits 2-7 are gated into the PSW bits 34-39. The MODAR trigger is also turned on in this cycle.

IE2

The GROUT and block T1M triggers are gated off in this cycle.

IEL

The MODAR trigger is gated off.

STORE MULTIPLE (STM)

From 1 to 16 general registers can be stored in successive memory locations. The first general register is specified by the R1 field, and the last general register is specified by the R3 field; R0 follows R15. The beginning storage address is specified by B2 + D2 and is incremented during the operation.

Condition Code: Unchanged.

Program Interruptions: Protection, addressing, and specification.

Details

Store multiple is executed jointly by the IE and E units. The IE unit handles the store requests and the delivery of the words in the general registers to the E unit. The E unit routes the words, one at a time, to the K register for delivery to storage.

The IE unit makes the store requests and sets the marks register. It also increments BR1 (initially set with the first GR location) to allow the proper general register to be sent to the E unit. The BR1 register is compared against IR2 (set with R3, the last GR location) to initiate an end to the IE unit operation.

The IE unit is kept synchronized to the transmission of words to storage by the accept pulse from BCU. The receipt of an accept pulse allows the IE unit to operate for two cycles, after which time another accept pulse is necessary to allow the IE unit to operate for two more cycles.

The E unit controls the routing of the general registers from the RBL to the left or right half of the K register. From the K register, the GRs are sent to storage in pairs (with the possible exception of the first and last GRs). The E unit keeps track of the transferred GRs by incrementing ER1 (the first GR) and comparing it against IR2 (the last GR, in the R3 field), A compare initiates an end to the E unit operation.

The E accept trigger (turned on by the accept pulse) is used to permit the E unit to operate for two uninterrupted cycles, after which it is again necessary to have the E accept trigger on.

An invalid address or one that specifies a protected area causes an interrupt at the end of the store multiple instruction. Execution of the instruction itself is unaffected, but stores are prevented when erroneous addresses are used.

As a service aid, the store multiple instruction can be used to store the content of any one general register into all locations of storage in a continuous, non-ending operation. To do this, the store multiple instruction must be manually executed (set in the data keys, then in the AB registers, etc.) with the enable storage ripple switch on. This switch prevents incrementing BR1 and ER1; therefore, if the first and last specified general registers (R1 and R3) are unlike, the store multiple instruction cannot make an equal comparison in its execution as long as the enable ripple mode switch is on. In addition, because continuous execution of the store

multiple instruction will soon be calling for locations outside the available storage, the enable ripple mode switch also prevents the invalid address signal from being generated by the BCU. To prevent SAP errors and to be able to store in protected locations, the storage protect key in the PSW must be 00.

The first storage address is calculated in T1, and is set into SAR and H registers at TN T2. IC fetches are blocked to prevent the ICR value from being gated into the incrementer, which is used by STM to update the storage addresses; T1 is blocked to prevent future address calculations to interfere with GR readouts; and T2 is blocked to protect the store address in SAR when the T1 block is removed.

The GROUT trigger is turned on at I to E transfer (the two execution units are started simultaneously). GROUT stays on long enough to gate the GRs selected by BR1 into RBL. Gating out R1 +1 is blocked.

Figure Reference: Figure 6352.

IE Unit

The controlling sequencers for the IE-unit operation are IE1, IE3, and two sequencers used especially for store multiple and load multiple, IM1 and IM2. The IM1 and IM2 sequencers alternate throughout the operation; IM1 is associated with the GRs that are stored in the left half of the double storage word, and IM2 is associated with the GRs in the right half of the double storage word. The IE1 sequencer is used at the beginning of the operation, and the IE3 sequencer is used at the end.

IE1/IM1

IE1 is always the first sequencer turned on, and IM1 or IM2 are turned on with it, depending on H bit 21. If H bit 21 is 0 (first GR goes in left half of double storage word), IM1 is turned on; if H bit 21 is 1 (first GR goes in right half of double storage word), IM2 is turned on.

In the IE1/IM1 cycle, the marks register four high-order bits (0-31) are set. In addition, if GR1 $\neq$ BR2 (these registers are equal in this cycle when only one GR is specified), BR1 is incremented. Setting the marks register and incrementing BR1 are IM1 functions and have nothing directly to do with IE1. However, IE1 is necessary to signify the first IE cycle and thus allow the IM1 functions; at all other times, these functions must wait for an accept pulse from BCU.

The incrementing of BR1 is effective in the next cycle (A clock). In this cycle, therefore, BR1 contains the address of the first GR sent to RBL. While GROUT controls the gating to RBL, the GR selection is controlled by BR1.

IE1/ IM2

This combination of sequencers defines the first I-execute cycle if H bit 21 is 1, as previously explained. If H bit 21 is 0, this cycle follows the IE1/IM1 cycle. Thus, IE1 is on for one or two cycles, depending on the status of H bit 21.

In the IE1/IM2 cycle, the MODAR trigger is set; a store request is made; the marks register is set for bits 32-63; and if BR1 $\neq$ IR2, BR1 is incremented to the next higher GR address. These are IM2 functions and IE1 is not involved. However, IE1 blocks incrementing the SAR and H registers, which is another IM2 function. Because the first store request is made in this cycle, the SAR register must remain undisturbed until the request is accepted.

The second GR (or first, if this is the first IE cycle), is gated to RBL, and will be stored in the right half of the double storage word. The incremented BR1 value is set in the register at the next A clock.

IM1

The IM1 cycle follows the IE1/IM2 cycle or the IM2 cycle. When only the IM1 sequencer is on, it must wait for the accept pulse from BCU to become effective. Until then, the IM1 sequencer defines idle cycles. The cycle in which accept (a B-B pulse) is received becomes a valid IM1 cycle. Two GRs have been delivered to RBL and taken from there by the E unit since the last accept pulse. The latched output of IM1 gates the turn-on of the store request trigger to maintain a constant store request.

In a valid IM1 cycle, the marks register is set for 0-31 bits, and if BR1 $\neq$ IR2, BR1 is incremented to send the next higher GR to RBL in the next cycle. Although the GR specified by BR1 is sent to RBL in all IM1 cycles (idle or not), the E unit does not accept RBL output until after the valid IM1 cycle.

IM2

The IM2 cycle follows a valid IM1 cycle. Because the last store request has been accepted (indicated by the IM2 sequencer being on), another store request is made; the SAR and H registers are incremented; and the marks register is set for bits 31-63, the positions that will be occupied by the GR sent to RBL in this cycle. In addition, if BR1 $\neq$ IR2, BR1 is incremented to send the next higher GR to RBL in the next cycle(s).

IE3/IM2

A compare equal of BR1 (R1 field) and IR2 (R3 field) during IE1/IM1 or IM1 starts this cycle. A compare equal during IE1/IM1 or IM1 indicates the last GR

has been gated into RBL, that it will occupy the left half of a double storage word, and that a store request for it has not been made if the compare occurred in IE1/IM1. The IE3/IM2 cycle, therefore, makes the store request and increments the SAR and H registers; the marks register has been set (for 0-31) in the previous cycle for the last GR. This cycle is followed by IE3/IM1.

The GROUT and block T1 triggers are turned off (a function of BR1 = IR2) at the beginning of this cycle because no other GRs are required for this operation.

IE3/IM1

A compare equal of BR1 (R1 field) and IR2 (R3 field) during IE1/IM2 starts this cycle. The preceding cycle (IE3/IM2) also starts this cycle.

In any case, the last GR has been sent to RBL and is stored in the right half of the double storage word. Because the IM2 sequencer was on in the preceding cycle, a store request for the last GR or pair of GRs has been made and the SAR and H registers have been incremented. The next E accept pulse sets the K register, containing the last GR, into the SBI latches. Therefore, the only function to be performed by the IE unit is to reset the GROUT and block T1M triggers (if not done in the previous cycle); and to reset the blocks to the T2 cycle and to IC fetches.

The turn-off of block T2M and block ICM triggers must wait until the accept pulse arrives from BCU in order to protect the SAR address.

IEL

This cycle follows IE3/IM1 after the BCU accept pulse arrives. The BCU accept pulse is also used to turn on ELC in the E unit, making IEL and ELC coincident. The MODAR trigger is gated off.

E Unit

The controlling sequencers for the E-unit portion of the operation are first FXP, store, and two sequencers used especially for store multiple and load multiple, EM1 and EM2. The EM1 and EM2 sequencers operate alternately to handle the delivery of the GRs taken from RBL into the left half and right half of the K register respectively, and to compare the R1 field against the R3 field, incrementing R1 when the two fields are unequal. First FXP is the beginning sequencer, and store follows an ER1-IR2 compare equal condition.

First FXP

This is the first cycle in the E-unit operation, and is coincident with the first IE cycle. First FXP is used to provide the RBL-M gate; thus, with GROUT (turned on at I to E transfer), first FXP completes the data path that sends the first GR to RBL and into the M register.

In the next cycle, the first GR is placed in the left or right half of the K register, and a second GR is placed in the M register.

The block PA trigger is gated on in this cycle to prevent any put-aways that are normally done during ELC.

EM1

The EM1 cycle delivers the GR in the M register, through the adder, into the left half of the K register; brings the next GR from RBL into M register; and increments ER1 if ER1 (R1 field) is not equal to IR2 (R3 field).

The EM1 functions take place provided the K register is either empty (following the first FXP), or not empty, but will be transferred to SBI (EBR pulse) in the same cycle. When EM1 follows the first FXP, the first GR is placed in the K register, and so no delay exists in completing EM1. When EM1 follows EM2, the K register contains two GRs (or one in the right half if EM2 followed the first FXP) that must be transferred to SBI before anything else can be placed into K register. In the latter case, the EM1 functions are held up until the E accept trigger is turned on (by the BCU accept pulse). The cycle in which E accept is on constitutes a valid EM1 cycle; until then, EM1 cycles are idle.

The idle EM1 cycles are those in which the EM1 trigger is on but the EM1 latch is not, since all EM1 functions are gated by its latched output. The valid EM1 cycles, therefore, are those in which the E accept trigger turns on the EM1 latch.

EM2

If the EM2 cycle follows the first FXP (H21 is 1), EM2 places the first GR in the right half of the K register. If EM2 follows EM1, it places the second of a pair of GRs in the right half of the K register.

The GR in the M register is routed through the adder with a R32 shift and placed in the right half of the K register. Another GR (if there is one) is taken from RBL and placed in the M register. If ER1 does not equal IR2, the ER1 register is incremented.

With ER1 ≠ IR2, the next cycle is EM1. The EM1 cycle repeats, as previously explained, waiting for the cycle that delivers the K register to the SBI.

Store

Store is an idle cycle, entered when ER1 = IR2 (R1 field matches the R3 field), and is repeated until a BCU accept pulse allows it to end and the next cycle to be ELC.

An equal compare occurs in the E unit during the cycle in which the last GR is put in the K register. The E unit operation ends when the accept pulse causes the K register content to be taken into storage.

An independent comparison of the R1 and R3 fields is made by the E unit because the IE unit compare equal condition occurs one cycle before the last GR is transferred to the K register in the E unit. The IE unit increments the R1 field (BR1) in the cycle in which any given GR is transferred through RBL to the M register; and the E unit increments the R1 field (ER1) in the cycle in which the same GR is transferred to the K register.

ELC

The ELC cycle occurs after the last BCU accept pulse is received. In this cycle, the last GR or pair of GRs in the K register is transferred to SBI, the block put-away trigger (turned on by the first FXP) is reset (next A pulse), and the MODAR trigger is reset (next A pulse).

LOAD MULTIPLE (LM)

A group of 1 to 16 storage words can be loaded into successive general registers. The first general register is specified by the R1 field, and the last general register is specified by the R3 field; R0 follows R15. The beginning storage address is specified by B2 + D2 and in incremented by 1 during the operation.

Condition Code: Unchanged.

Program Interruptions: Addressing, specification.

Details

Execution of load multiple is accomplished by both the IE and E units; the IE unit handles the fetch requests and the E unit handles the delivery of the storage words to the general registers.

The IE unit makes fetch requests and increments the storage addresses in SAR and H registers; however, the first fetch request and address incrementing is done by the I unit. The IE unit keeps track of the words fetched from storage by incrementing BR1 (initially set with the first GR location) and comparing it with IR2 (set with R3, the last GR location). When the two registers compare equal, IE unit operation ends.

The accept pulse from BCU keeps the IE unit properly synchronized to the words coming back from storage. Except when the BR1 and IR2 fields compare equal, every accept pulse causes the IE unit to proceed with two useful cycles, in which the BR1 field is incremented twice and the next storage address is calculated. If there is any delay in receiving the accept pulse, the IE unit is kept idle for the duration of the delay except for maintaining a fetch request.

The E unit takes the words fetched into the J register and passes them through the adder into the left half of the K register. From the K registers, the words are placed in the general registers specified by ER1. Initially set with R1, the first GR location, ER1 is incremented +1 by the E unit until it matches IR2 (the last GR location). When the final word has been placed in the last specified general register, the E unit operation ends, completing the instruction (the IE unit operation ends sooner, as it has only to wait for the accept signal for the last requested storage word).

As in all instructions requiring a fetch, the E unit uses the J loaded signal to allow it to process word(s) fetched to the J register. When J becomes loaded, the E unit progresses for two cycles in which the two words fetched are placed in the proper general registers, and in which ER1 is incremented twice. If the next J loaded signal does not occur immediately, the E unit is kept idle until the J register is again loaded.

Detection by the BCU of an invalid (non-existent) storage address or one that specifies a protected area (SAP fetch) causes the E unit operation to end early to prevent loading the general registers with invalid data. In addition, the proper error triggers are turned on to cause and identify an interrupt when the I unit operation ends.

As a service aid, the load multiple instruction can be used to load all words in storage into any one general register in a continuous, non-ending operation. To do this, the load multiple instruction must be manually executed (set in the data keys, then in the AB registers, etc.) with the enable storage ripple switch on. This switch prevents incrementing ER1 and BR1; therefore, if the first and last specified general registers (R1 and R3) are unlike, the load multiple instruction cannot make an equal comparison in its execution as long as the enable ripple mode switch is on. In addition, because continuous execution of this instruction would soon be calling for locations outside the available storage, the enable

ripple mode switch also prevents the invalid address signal from being generated by the BCU. To prevent SAP fetch errors and to be able to read out of protected storage locations, the storage protect key in the PSW must be 00.

During T1 and T2, the first storage address is calculated (B2 + D2) and the first fetch request made. When accept is received, E go and I go are generated, and the first storage address, incremented by 1, is placed in SAR and H registers. T1 is blocked (block T1M) to prevent the next instruction time from using a general register in an address calculation until all general registers have been loaded by this instruction. IC fetches are also blocked (block ICM) to keep the ICR value out of the incrementer and to keep the I unit from making competing fetch requests.

Figure Reference: Figure 6353 and Figure 6352 (the logic expressions).

IE Unit

The controlling sequencers used by the IE unit for load multiple are IE1, IM1, and IM2. The IE1 sequencer is used for the first one or two execution cycles, and IM1 and IM2 are used alternately as in store multiple.

When the BCU responds to the I unit's fetch request with an accept signal, I go is generated, which turns on IE1 along with IM1 or IM2, depending on whether the first word to be loaded is located in the the left half or right half of the double storage word.

IE1/IM1

IE1 is always the first sequencer turned on, along with either IM1 or IM2, depending on H bit 21. If H bit 21 is 0 (first word located in left half of double storage word), IM1 is used; if H bit 21 is 1 (first word located in right half of double storage word), IM2 is used.

In IE1/IM2, the beginning general register address in BR1 is incremented +1 if it is not equal to IR2, the address of the last general register. If the BR1 and IR2 registers are equal, BR1 is not incremented, the block IC trigger is turned off, and the IE unit operation ends by turning on IEL.

The functions mentioned are handled by IM1 alone, and not by IE1. However, IE1 is on to prevent another IM1 function, that of making a fetch request (by the latched output of IM1.) If no effort were made to prevent this, and the BR1-IR2 fields were equal, an unnecessary fetch can occur (equal fields in this cycle means that only one word needs to be loaded and is contained in the storage word already called for by the I unit.)

Another reason for IE1 is to make IM1 independent of the accept pulse for its completion. Normally, IM1 must wait for accept before it can increment BR1 or proceed to the next cycle.

IE1/IM2

This combination of sequencers defines the first execute cycle if H bit 21 is 1, as explained earlier. If H bit 21 is 0, this cycle follows the IE1/IM1 cycle.

In IE1/IM2, the I-unit operation can end if the BR1 and IR2 fields are equal. The block ICM trigger is turned off and the next cycle is IEL. If the BR1 and IR2 fields are unequal, BR1 is incremented +1 and a fetch request is made by the latched output of IM2.

Here again, the functions mentioned are handled by IM2 only and not by IE1. However, IE1 prevents the incrementing of SAR and H registers, another of IM2's functions. Incrementing SAR and H here would by excessive because these registers were already incremented by the I unit (T2) for a possible second fetch request.

IM1

The IM1 cycle follows the IE1/IM2 cycle or the IM2 cycle. The latched output of IM1 maintains the fetch request made originally by the latched output of IM2. The IM1 sequencer stays on until an accept is received. Until this signal is sent by the BCU, IM1 does nothing useful other than maintaining the fetch request. The fetch request (IE fetch to J) must be maintained by the IE unit as there is no intervening trigger or latch between it and the BCU to remember the request.

Once the accept signal arrives, the IEL sequencer is turned on next if there is a match between the BR1 and IR2 fields. If there is no match, BR1 is incremented +1 and the IM2 sequencer is turned on.

IM2

IM2 always follows the IM1 cycle if in IM1 there was no match between BR1 and IR2. The SAR and H registers are incremented +1 by IM2. BR1 is also incremented if it is not yet equal to IR2, and the next cycle is IM1. If BR1 is equal to IR2, the block ICM trigger is turned off and IEL is turned on.

Because the SAR and H registers are incremented unconditionally in IM2, they will be incremented one time more than necessary if BR1 and IR2 are equal in this cycle.

IEL

The IEL cycle follows the cycle in which equal fields are detected in BR1 and IR2 (BR1 Eq IR2 latch on). Normally, IEL occurs in the IE unit before ELC in the E unit. This is because the last accept signal

occurs before the last J loaded signal. However, in case an invalid storage address or an address that is in a protected storage location was used for one of the fetches, ELC is forced in the E unit and the block PA trigger is turned on to prevent further put-aways. Thus, in case one of these errors occurs, the instruction is ended as soon as IEL occurs in the IE unit, whose operation is unaffected by the errors.

E Unit

The E-unit operation is controlled by first FXP, EM1, and EM2. First FXP is the first sequencer, turned on by E go, and is followed by the EM1-EM2 sequencers which are turned on alternately.

First FXP

First FXP delivers the first word loaded into J through the adder to K0-31. If the first word is in the left half of the double storage word (H bit 21 is 0), J0-31 is gated to the adder; if the first word is in the right half of the double storage word (H bit 21 is 1), J32-63 is gated to the adder. E go turns on the first FXP trigger and J loaded turns on the first FXP latch.

If the first fetch request was made for an invalid (non-existent) storage address, the address invalid trigger is turned on at the same time as the J loaded trigger. Similarly, if the first fetch request was made for a protected storage location, the storage address protect (SAP) trigger is turned on along with the J loaded trigger. Either of these error triggers coming on forces the block PA and ELC triggers to prevent a put-away and end the E-unit operation. In addition, depending on which of the error triggers is turned on, the SAP interrupt or address interrupt trigger is turned on to cause and identify an interrupt when both the E and IE units become not busy.

EM1

This cycle follows EM2, or first FXP if H bit 21 is 1. In EM1, the word in K0-31 is set into the general register specified by ER1, and the MODAR trigger is turned on. If ER1 equals IR2, the IM2 latch is turned on immediately, T1 is unblocked, and the block PA and ELC triggers are turned on in the next cycle. If ER1 does not equal IR2, the EM1 latch waits for the J loaded trigger (accept trigger in single cycle) before setting the word from J0-31 to K0-31 and incrementing ER1 +1. The next cycle is EM2.

EM2

EM2 gates the word in J0-31 (put there in first FXP or EM1) through the adder to K0-31. Also, the word presently in K0-31 is set in the general register specified by ER1. The MODAR trigger is turned on, and if ER1 is not equal to IR2, ER1 is incremented +1 and the next cycle is EM1. If ER1 and IR2 are equal, the block T1M trigger is turned off, and the block PA and ELC triggers are turned on.

ELC

Normally, ELC follows an EM1 or EM2 cycle in which the ER1 and IR2 fields are found equal. The block PA trigger is on at the beginning of this cycle to prevent the normal ELC put-away; the last word enters the last specified general register in the preceding EM1 or EM2.

If the invalid address trigger or SAP trigger is turned on in first FXP or EM1, ELC is forced by the error condition. An interrupt is taken when the IE unit completes its end of the operation, which is unaffected by either the SAP or invalid address error.

START I/O (SIO)

A write, read, read backward, control or sense operation is initiated at the addressed I/O device and subchannel. Bits 21-31 of B1 + D1 identify the channel, subchannel, and I/O device. The start I/O instruction can be executed only in the supervisory state (PSW 15 is 0).

Condition Code:

0 I/O operation initiated and channel proceeding with its execution
1 CSW stored
2 Channel or subchannel busy
3 Not operational

Program Interruptions: Privileged operation.

Details

The details regarding the CPU's involvement is the same for all four I/O instructions. See the Details section of Halt I/O.

TEST I/O (TIO)

The state of the addressed channel, subchannel, and device is indicated by setting the condition code in the PSW and, under certain conditions, by storing the CSW. Pending interruption conditions may be cleared. Test I/O can be executed only in the supervisory state (PSW 15 is 0).

Bits 21-31 of B1 + D1 identify the channel, subchannel, and I/O device.

Condition Code:

0 Available
1 CSW stored
2 Channel or subchannel busy
3 Not operational

Program Interruptions: Privileged operation.

Details

The details regarding the CPU's involvement is the same for all four I/O instructions. See the Details section of Halt I/O.

TEST CHANNEL (TCH)

The condition code is set to indicate the state of the addressed channel. The channel is unaffected. Bits 21-31 of B1 + D1 identify the channel. The condition code is set to indicate the state of the channel addressed by bits 21-23 of B1 + D1. Test channel can be executed only in the supervisory state (PSW 15 is 0).

Condition Code:

0 Channel available
1 Interruption pending
2 Channel operating in burst mode
3 Not operational

Program Interruptions: Privileged operation.

Details

The details regarding the CPU's involvement is the same for all four I/O instructions. See the Details section of Halt I/O.

HALT I/O (HIO)

The operation being executed by the addressed subchannel or channel is terminated. Bits 21-31 of B1 + D1 identify the I/O device, and channel or subchannel. Halt I/O can be executed only in the supervisory state (PSW 15 is 0).

Condition Code:

0 Channel and subchannel not working
1 CSW stored
2 Burst operation terminated
3 Not operational

Program Interruptions: Privileged operation.

Details

The start I/O, test I/O, test channel, and halt I/O instructions are executed by the IE unit. The bits 21-31 of B1 + D1 and a line specifying one of the four instructions are sent to all channels beginning at TN T2. Later in the execution time, when it is assured that these lines are settled, a select line is sent to one of the channels to enable it to accept these earlier signals. CPU then waits for a response from the selected channel; when it is received, CPU sets the condition code, resets the channel interrupt priority circuits, and ends the instruction.

If the instruction specified channel 7, no channel selection is made (the seven channels are addressed 0-6). CPU recognizes the invalid address 7 and generates its own release. The condition code is set to 3, and the invalid channel specification trigger is set to cause an interrupt at the end of the instruction.

If the channel address is valid but the channel is in test mode, or is not connected in the system, or its meter is disabled, CPU again generates its own release, sets the condition code to 3, and ends the instruction.

Bits 16-23 are sent from the H register, with a parity bit, over the unit address bus to all channels. The single instruction line sent to all channels comes from the BOP register. To protect the content of the two registers until a release is received or generated, T1 is blocked.

Figure Reference: Figure 6354.

SET STORAGE KEY (SSK)

- Function of SPF is to prevent storing into a location inadvertently.
- SPF holds a key for each 256-word block of storage.
- Set key instruction allows the programmer to store a key in the SPF.

This RR-format, a privileged instruction, sets a storage protection key into SPF storage.

The SPF storage holds a four-bit key and a read-protect bit (plus a parity-bit) for each block of 256 words (storage words). The 256 storage-word (2048-byte) blocks are pointed to by address bits 12-0. In other words, address bit 12 changes once for every 2048 consecutive byte addresses. There is one SPF storage in each 2365 Processor Storage unit. This SPF serves both the even and the odd high-speed storage (HSS) within the 2365. The number of SP locations within a 2365 and the address bits used to address SP storage depend on the system storage configuration:

Model	SP Model	Number of SP Locations	Address Bits Sent to SP	Comments
H75	III	128	6-12	HSS 2-way Interleaved
I75	V	256	5-12	HSS 4-way Interleaved
J75	V	256	5-12	HSS 4-way Interleaved

The function of the SPF is to prevent using a storage location inadvertently. Whenever a storage operation is called for, the SPF fetches the prestored SP word corresponding to the incoming address. The fetched key is compared with the key

furnished by the storage user on all store operations and on fetches if the read-protect bit is a logical 1. On CPU initiated storage operations, the key is supplied from the PSW, bits 8-11; for channel initiated operations, the key is supplied by the channel, which originally got the key from from a channel command word (CCW). If there is a bit-by-bit match of the two keys, or if the key supplied by the storage user is all zeros, the operation is allowed to proceed. If this condition is not met, the SPF signals an error to the selected HSS and signals a storage address protect (SAP) error to the BCU. On store operations, the selected HSS is cancelled causing the addressed location to be rewritten without change (regenerated). On fetch operations, data output from storage is blocked; the SBO contains all zeros (with good parity).

Data Flow

- Bits 8-31 of general register R2, are routed through the AA and set into SAR.
- General register R1 is placed on GBL, where bits 24-31 are picked-off and passed through the BCU key gate.
- BCU generates a parity bit and sends five bits plus parity to the SPF.

The set storage key instruction is the means by which a configuration of key bits for a block of storage is set into the SPF. On this instruction, SPF storage is addressed by the contents of general register R2 and the key and read-protect bit set into SPF are taken from bits 24-28 of general register R1.

General register R2, bits 8-31, is routed through the AA and set into SAR 0-23 (Figure 6355). As on a CPU fetch or store, the BCU sends bits 6-19 of SAR (H75) or 5-18 (I75, J75) to storage. SAR bits 6-12 (H75) or 5-12 (I75, J75) are routed to the SPF to address the SP location to be stored.

General register R1 is routed on GBL to the AA input OR, where bits 24-31 are picked-off and sent through the key gate in the BCU. The BCU generates a parity-bit for the key and sends the five bits (plus parity) through the key OR to the SPF.

Control

- GR R2 is set into SAR and H during I time.
- GROUT is set to gate out GR R1 during E time.
- IE1 makes fetch request.
- Accept turns off IE1 and turns on IE2.
- BCU treats fetch as if it were a store.
- A HSS is selected, but cancelled.

Sequencers used for the set storage key instruction are shown in Figure 6355. During I time, GR R2 is gated to the AA and the output of the AA is set into SAR and H. General register out (GROUT) is turned on to gate GR R1 through the AA input OR to the BCU key gate.

The execution is accomplished by I execute and the IE1 sequencer sends a fetch request to the BCU. A set key line is also sent to the BCU. IE1 repeats until the BCU responds with accept. After accept, the IE2 and IEL sequencers finish the operation.

The set key line alters the operation of the BCU. The fetch is treated as a store operation insofar as returning data and handling errors. A return address register is not set even though return to J is active. A set key line is sent to storage. This line causes the selected HSS to cancel its operation and tells the SPF to store the incoming key. The BCU generates a parity bit for the incoming key and gates the five bits plus parity to the SPF. The BCU actually receives a full byte plus a parity bit from GR R1. Parity sent to the SP unit is generated by examining bits 29-31 (the unused bits in the byte), then changing the byte parity bit if an odd number of 1 bits are being removed from the byte. This parity generation scheme prevents correcting bad parity. If the parity of the byte received from GR R1 is bad, BCU sends bad parity to the SPF, where a parity error is generated.

INSERT STORAGE KEY (ISK)

- Instruction fetches an SPF key.
- Fetched key is set into GR R1, bits 24-28.

This RR format, a privileged instruction, fetches the addressed SPF key and sets it into GR R1 bits 24-28. The insert storage key instruction is the means by which a programmer can examine a previously-stored protection key for a particular block of storage.

Data Flow

- GR R2, the SPF address, is routed through the AA and set into SAR.
- SAR is routed through the address OR to storage.
- SPF delivers the fetched key to the BCU key buffer register (KBR).

- GR Rl is routed through RBL to M, to AM, and into K.
- The fetched key is routed from the KBR through the AOE mask and into K24-31.
- K is put-away in GR Rl.
- Bits 24-28 of GR Rl contain the fetched SP word. Bits 29-31 of GR Rl contain zeros.

General register R2, bits 8-31, is routed through the AA and set into SAR 0-23 (Figure 6356). As on any CPU fetch or store, the BCU sends bits 6-19 (H75) or 5-18 (I75 and H75) of SAR to storage. SAR bits 6-12 (H75) or 5-12 (I75 and J75) are routed to SPF to address the key to be fetched.

The SPF delivers the addressed key to the BCU key buffer register. The key (5 bits plus a parity-bit) is routed from the key buffer to the AOE mask where three zeros are added to make a full byte.

Meanwhile, general register Rl is routed through RBL to M and from M through the main adder to K. The AOE mask byte, which contains the fetched key, is set into K24-31. This byte replaces the corresponding byte from GR Rl. Bits 0-31 of K are then set back into GR Rl to complete the instruction.

At the end of the instruction, bits 24-28 of GR Rl contain fetched key. Bits 29-31 of GR Rl contain zeros, and the remainder of GR Rl is unchanged.

Control

- Insert key executed by E-and IE-unit sequencers.
- IEl makes the fetch request.
- Accept turns off IE1 and turns on IE2.
- First FXP routes GR Rl to AM.
- Hwd sets AOB to K and KBR to AOE mask.
- ELC does Rl put-away.

Sequencers used for the insert storage key instruction are shown in Figure 6356. During I time, GR R2 is gated to the AA and the output of AA is set into SAR and H. General register Rl is gated out to the RBL and both I- and E- execution units are started.

The IEl sequencer brings up I fetch request and insert key lines to the BCU. IEl cycles repeat until BCU responds with accept. On accept, the I- execute unit goes into the IEL cycle, then takes no further part in the execution of this instruction.

The first fixed-point (FXP) sequencer gates RBL to M and M to AMTC. First FXP cycles repeat until advance returns from SPF. The SPF advance occurs about 250 nanoseconds after the BCU generates select. Once advance arrives, the first FXP sequencer is turned off and the halfword logical sequencer is turned on. During the halfword logical cycle, the main adder output (AOB) is set into K and the BCU key buffer register is routed to the AOE mask.

Following the halfword logical cycle, the ELC sequencer turns on to set the AOE latch into K24-31 and to put 0-31 of K back into GR Rl.

The BCU handles the insert storage key instruction almost identically to the way it handles the set storage key instruction. The insert storage key line causes the BCU to treat the fetch as a store operation; no return address positions are set even though return to J is up and any errors detected during the operation are treated as if they occurred during a CPU store.

The insert key line to the selected HSS causes a cancel and insert to SPF causes fetching of the addressed key. The SPF advance gates the fetched key into the key buffer and also signals the E unit that it can proceed from first FXP cycles to the halfword logical cycle.

DIAGNOSE

The purpose of the diagnose instruction is to set the MCW register, the positions of which subsequently control various CPU and channel functions. These functions are mainly the forcing of errors so that the error checking stations in the CPU and the channels can be tested.

The diagnose instruction fetches a double word from storage and sets the left half of it into the MCW register (Figure 6357).

The diagnose instruction is executed by the IE and D sequencers (Figure 6357). During I time, the storage address is calculated and set into SAR. IEl makes the fetch request and maintains the request until the BCU responds with accept. The IEl cycles also send a diagnose signal to the BCU that sets the diagnose position of one of the return address registers.

After BCU generates accept, no sequencers are on until the advance pulse from the selected storage samples the return address register and generates diagnose select. The diagnose select signal (delayed approximately 150 nanoseconds) gates SBO 0-31 into the MCW register, and at the same time turns on sequencer Dl.

If the cycle-count feature of the MCW control is not enabled, sequencers Dl, D2, and D3 serve no useful purpose; D3 generates a proceed signal that turns on IE3, allowing the instruction execution to continue.

If the cycle-count is enabled, the CPU is allowed to run, following the proceed signal, only the number

of cycles specified by the count in the MCW count field. Sequencers D1, D2, and D3 monitor the MCW counter so that when the count is reduced to zero, the controlled clock will be stopped and a log-out taken.

INTRODUCTION

- A branch is a departure from sequential instruction processing.
- The preparation and execution units perform the same functions for branch instructions as they do for all other instructions.
- The branch unit controls the fetching of instructions from the branch address and on a successful branch starts processing from the branch address.

The branch instructions have much in common with all other instructions. They require that storage addresses be computed, that fetch requests be made, that registers be delivered to the E unit, and that arithmetic be done. These are all done by the preparation unit or the E unit as they are for other instructions. The branch instructions differ in one important way; based on a decision made during their execution, the branch instructions may or may not require a departure from normal sequential instruction processing.

On an unsuccessful branch, as for all non-branching instructions, the instruction at the next higher storage address is the next to be processed and has normally been prefetched to the instruction buffers by the IC controls. A successful branch, however, requires that the next instruction processed be from a different storage address. The purpose of the branch unit is to make this alternate instruction available as early as possible without slowing normal processing if the branch is unsuccessful.

The branch unit controls the fetch requests for instructions from the branch address, handles the returning fetches so that they are not delivered to the instruction buffers unless the branch is successful, and on successful branches, changes the GSR and the ICR so that normal sequential instruction processing starts from the branch address.

UNITS OTHER THAN BRANCH UNIT

- The I unit computes the branch address and starts the executing units.
- The E unit performs arithmetic tests and moves data.
- The IE unit performs a no operation on BCR if R2 = 0.

Figure 3 lists all branch instructions and shows the I and the E unit functions performed for each. I unit computes the branch address and starts the executing units. E unit does the arithmetic to determine success and moves data.

In Figure 3 note that for RR branch instructions with R2 equal to zero, the branch unit is not started; these instructions contain no branch address and cannot result in a branch. BALR and BCTR with R2 zero are used only to accomplish their specified data movements. On BCR with R2 equal zero, the IE unit performs a two cycle no operation, no branch and no data movement result.

Note that on six of the nine branch instructions, the E unit and the branch unit both take part in the execution. The E unit does arithmetic and moves data and the branch unit controls the fetching of instructions from the branch address and starts the processing of either the branch instruction or the next sequential instruction, depending on the success of the branch.

Only the execute and the branch-on-condition instructions cause the branch unit to perform the execution alone. On BC and BCR, the success of the branch is determined by testing the CC portion of the PSW for conditions set up on a previous instruction. On execute, the branch is always successful, however, only one instruction is executed at the branch address.

Figure 4 shows the units that operate on the different branch instructions and the sequencers that control their operation.

Note that branch operation (Br Op), the first branch unit sequencer, is set at TN T2 one cycle earlier than the first cycle sequencers for other execution units.

BRANCH UNIT

- Uses circuits and devices in the same way as other units of the 2075.
  1. Most of the operation is automatic for all branches.
  2. Sequencers are used to control events that must be ordered in time.
  3. Memorized triggers are used to remember facts that are necessary for the control of asynchronous events.
  4. Trigger-latch pairs are used.
  5. The flush path property of the PH is used.
- Uses three sequencers.

Instruction	Format	Branch Address	Executing Units			Branch Condition	Data Change
			BR	E	IE		
BALR	RR	R2	R2≠0	X		Uncond R2≠0	RH PSW to R1
BAL	RX	X2+B2+D2	X	X		Uncond	RH PSW to R1
EX	RX	X2+B2+D2	X			Uncond (1 Inst)	Subject instruction modified BOP R1 (24-31) OR'ed to IOP (8-15) on T1 of subject
BCR	RR	R2	R2≠0		R2=0 (No Op)	CR ≅ M (8-11 IOP) M = 1111 No Op if M = 0000	none
BC	RX	X2+B2+D2	X			CR ≅ M (8-11 IOP) M = 1111 No Op if M = 0000	none
BCTR	RR	R2	R2≠0	X		R1 = 1 if R2≠0	(Content of R1) -1 to R1
BCT	RX	X2+B2+D2	X	X		R1 = 1	(Content of R1) -1 to R1
BXH	RS	B2+D2	X	X		R1+R3>R3 or (R3+1) which ever is odd	(R1+R3) to R1
BXLE	RS	B2+D2	X	X		R1+R3≤R3 or (R3+1) which ever is odd	(R1+R3) to R1

FIGURE 3. BRANCH INSTRUCTION DIFFERENCES

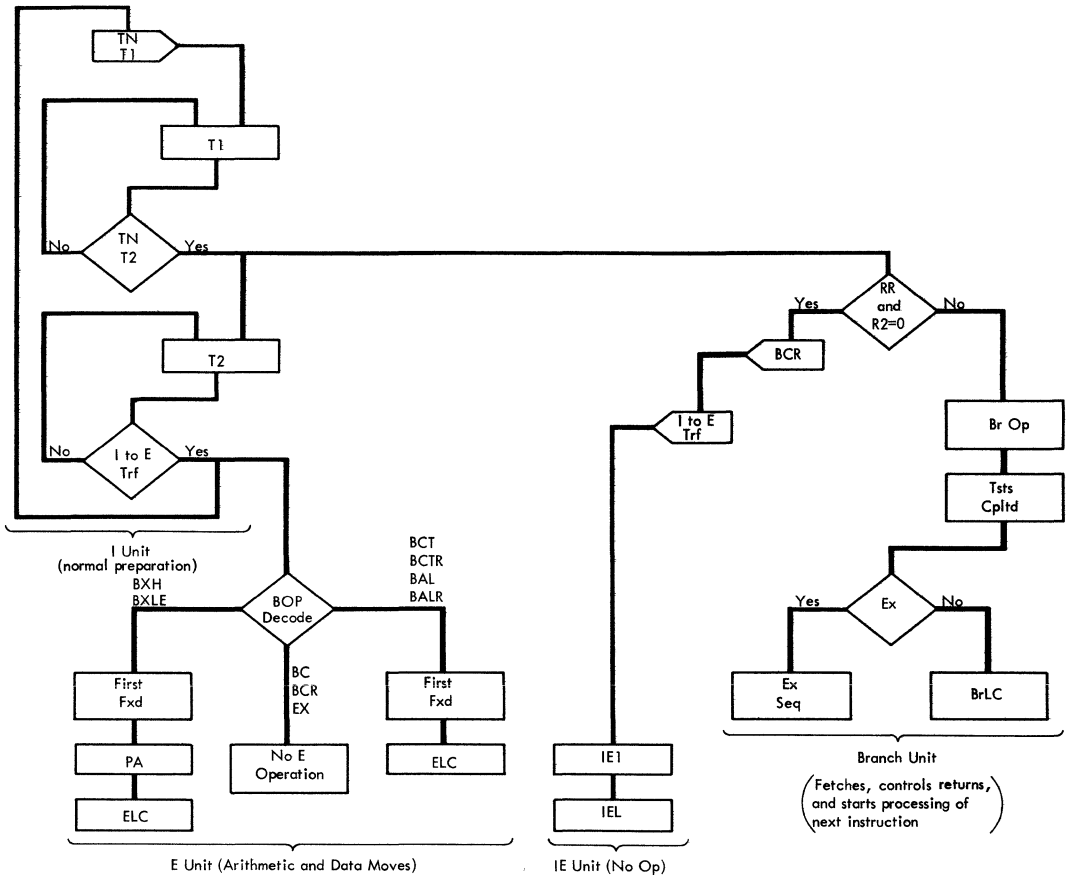


FIGURE 4. BRANCH INSTRUCTIONS, MAJOR CONTROL FLOW, UNITS AND SEQUENCERS

- Makes the branch and branch + 1 fetches conditionally.
- Buffers the branch fetch in the J register if it returns before success is determined.
- Examines and remembers results of tests so that returning fetches may be delivered to the instruction buffer or cancelled as required.
- On successful branches (except execute), sets the branch address to GSR and ICR.

Since RR branch instructions with R2 equal to zero never branch, the branch unit is not started. For all other branch instructions, the branch unit is started and must run its course. The branch unit is started by turning on branch operation at TN T2. Branch operation is always followed by tests complete and branch LC (or execute sequence for the execute instruction). At the same time that branch operation is turned on, OPF is turned on and the branch fetch is made. This fetch must be accepted before the I to E transfer can be made. At I to E transfer, branch + 1E is set and the branch + 1 fetch is made with two exceptions. On branch on condition instructions Br + 1E is not turned on and the fetch is not made unless the branch is successful. On Branch on index instructions Br + 1E is turned on at I to E transfer but the fetch is delayed one cycle. Figure 5 shows in heavy outline those functions that are automatic to any operation of the branch unit.

The block third outstanding fetch logic is part of the branch unit and always operates to delay any fetch to A or B if two previous fetches to either A or B have been accepted and not returned. This delay is necessary because the logic that remembers if such fetches are to be used or ignored upon return keeps track of only two such fetches.

Two blocks in light outline are closely related to the heavy outline or automatic portion of the figure. Branch M and branch +1M are turned on by the accepts for their respective fetches. Since the branch fetch must always be accepted before I to E transfer, branch M comes on whenever the branch unit operates. The branch +1 fetch, however, is dropped if it has not been accepted by the time that a branch is determined unsuccessful; therefore, branch +1M does not always follow branch +1E.

The remainder of the light outlined portion of Figure 5 is dependent on the success of the branch and conditions relating to returning fetches.

Branch success is set at tests complete only if the conditions for branching are met. With tests complete on, the condition of branch success determines if the branch address is set to GSR and ICR or not.

Sel A and Sel B are developed to set the A and B registers on successful branches. ICAM or ICBM are turned on to remember that a branch +1 fetch for a successful branch has been accepted and not returned.

The branch cancel triggers, Br CA1, Br CA2, Br CB1, and Br CB2, are set to remember that branch or branch +1 fetches for unsuccessful branches have been accepted and not yet returned. Their use enables an unsuccessful branch to be terminated and the next instruction started before the unneeded fetches are returned.

The upper right hand portion of Figure 5 shows the data paths used by returning branch fetches. Since the branch fetch is made at TN T2 and I to E transfer may be held up by a busy execution unit, the branch fetch may return before the success of the branch is determined. Under these conditions, the returning fetch is buffered in the J register and not set to A or B until tests complete and then only if the branch is successful. The branch +1 fetch request is made at I to E transfer or later and can never return before success is determined.

THEORY OF OPERATION

The processing of branch instructions is covered in two sets of figures. The first, a set of simplified logic diagrams, shows how the important branch unit functions are accomplished. The second, a set of flow charts, shows the cycle by cycle performance of all branch instructions for both preparation and execution.

BRANCH UNIT OPERATION

Branch Unit (5525)
Sequencers (5526)
Branch Fetch (5527)
Branch +1 Fetch (5528)
Branch +1 Fetch Address (5529)
Branch Successful (5530)
Set A/B Reg (5531)
TON ICAM (5532)
Branch Address to GSR and ICR (5533)
Cancel Triggers and Block
Third Outstanding Fetch (5534)

PREPARATION AND EXECUTION OF BRANCH INSTRUCTION

EX, BC, BCR (R2 $\neq$ 0); Preparation and Execution(6375)
BCR, (R2 = 0); Preparation and Execution (6376)
BAL, BALR; Preparation and Execution (6377)
BCT, BCTR; Preparation and Execution (6378)
BXH, BXLE; Preparation and Execution (6379)

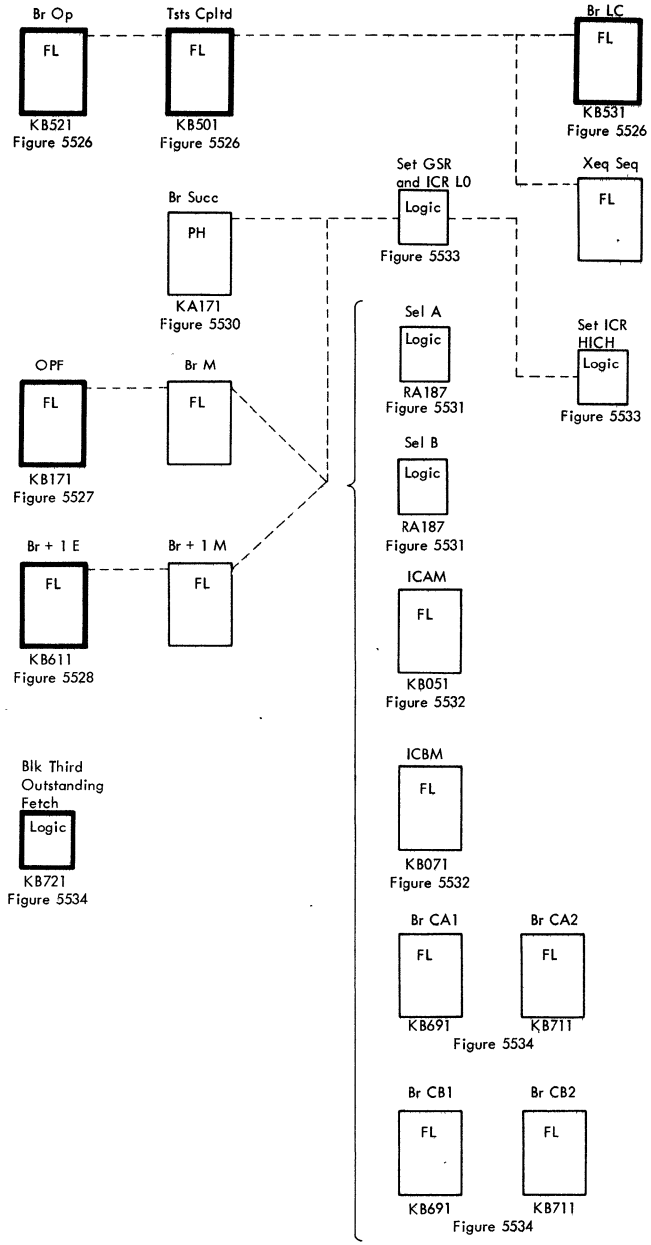
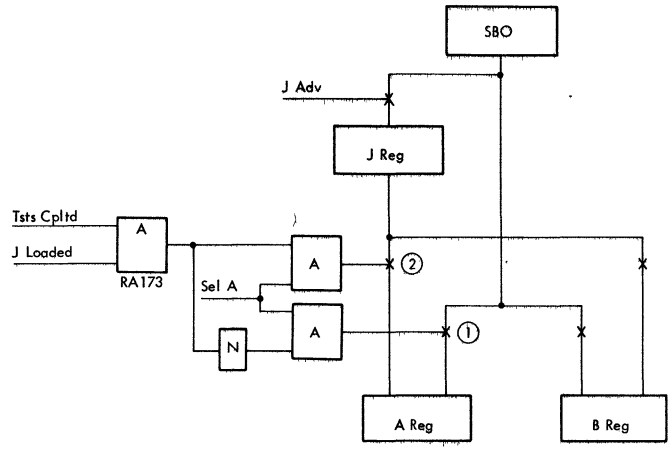


FIGURE 5. BRANCH INSTRUCTION DIFFERENCES



(1) Generally, Sel A/B sets A/B from SBO

(2) If Tsts cpltd and J loaded are on when Sel A/B is developed, instructions from the branch address already buffered in J reg are set to A/B.

Automatic Functions (heavy outline)

Branch Operation (Br Op) — First Sequencer, controls making of Br and Br + 1 fetches and marks operation as branch.

Tests Complete (Tsts Cpltd) — Second sequencer, times the use of Br SUCC with the completion of the test.

Branch Last Cycle (BrLC) — Last sequencer, ends operation and times setting of branch address to ICR on a successful branch. Exception is execute instruction when execute sequence ends the operation.

Operand Fetch (OPF) — Makes fetch for instruction at branch address.

Branch + 1 Execute (Br + 1E) — Makes fetch for next higher storage address.

Blk Third Outstanding Fetch — Logic keeps third fetch to A or B from being made if their are two accepted and not returned fetches to either A or B

Functions Dependent on Success

Branch Succ — Set if conditions for branching are met, controls returning fetches.

Set GSR and ICR — Logic sets branch address to GSR and ICR on successful branch.

Sel A/B — Set branch instructions to A and B on successful branch.

ICBM-ICAM — Used to remember that a Br + 1 fetch for successful branch is outstanding.

Br CA 1/2; Br CB 1/2 — Used to remember outstanding fetches for unsuccessful branches.

INTRODUCTION

- Allows a wide range of magnitude.
- Uses long and short operands.
- Computer keeps track of the decimal point.

The floating-point arithmetic feature enables computations to be performed using operands with a wide range of magnitude. The operands used in floating-point operations are in scientific notation. Both the number or fraction (mantissa) and its exponent (characteristic) are processed by the computer. The decimal point of a factor is initially located and the computer keeps track of it. Thus, the operator or programmer is not concerned with adding zeros to the answer obtained by the computer.

The floating-point arithmetic feature provides for addition, subtraction, comparing, division, halving, loading, sign control, multiplying and storing of short or long operands. Short operands generally require less storage space than long operands. On the other hand, long operands provide greater accuracy in a calculation.

NUMBER SYSTEMS

For a thorough understanding of the floating-point feature, you should be familiar with the binary and hexidecimal number systems. Figure 6 shows the symbols used in these systems and the decimal equivalents; Figure 7 shows the add-subtract chart and the multiplication-division chart. Arithmetic in these systems and conversion from one to the other are explained in an IBM Student Text, Number Systems, Form C20-1618.

Decimal	Hexadecimal	Binary	Decimal	Hexadecimal	Binary
0	0	0000	12	C	1100
1	1	0001	13	D	1101
2	2	0010	14	E	1110
3	3	0011	15	F	1111
4	4	0100	16	10	10000
5	5	0101	17	11	10001
6	6	0110	18	12	10010
7	7	0111	19	13	10011
8	8	1000	20	14	10100
9	9	1001	21	15	10101
10	A	1010	22	16	10110
11	B	1011	23	17	10111

FIGURE 6. DECIMAL, HEXADECIMAL, AND BINARY NOTATION

Instruction Formats

- RR and RX formats (Figures 8 and 9).
- Registers addressed must be an even address.

All floating-point instructions are either register to register (RR format) or storage to register (RX format).

The RR format specifies the operation code (bits 0-7), the first operand (bits 8-11) specifies a floating-point register, and the second operand (bits 12-15) specifies the second floating-point register that is taking part in the operation. The same register may be specified for the first and second operand.

The RX format specifies the operation code (bits 0-7) and two operands. The first operand (bits 8-11) specifies a floating-point register, and the second operand (bits 12-15, 16-19, and 20-31) is made up of:

1. Index (X2): The index is a 24-bit number contained in a general register. The general register is designated by bits 12-15 of the instruction.
2. Base Address (B2): The base address is a 24-bit number contained in a general register. The general register is designated by bits 16-19 of the instruction.
3. Displacement (D2): The displacement is a 12-bit number contained in bits 20-31 of the instruction.

The second operand is from the storage location specified by the effective address. The effective address is obtained by adding the contents of the index register (X2) (specified by bits 12-15 of the instruction word) to the contents of the base register (B2) (specified by bits 16-19 of the instruction word) and the contents of instruction word bits 20-31. The general registers are 24-bit positive binary integers, having no sign position. The displacement (D2) is treated as a 12-bit positive binary integer. The three binary integers are added together as 24-bit binary numbers, overflow is ignored.

The X2 and B2 fields may contain zeros. A zero indicates the absence of the corresponding address component; therefore, a base or index tag of zero indicates that a zero quantity is used to form the effective address. In an instruction specifying no base or index register, the effective address is specified by the D2 field (bits 20-31) of the instruction word.

The storage address of the second operand should designate word boundaries for short operands and double word boundaries for long operands; otherwise, a specification exception is recognized and a program interruption is caused.

The registers addressed by R1 and R2 fields should be 0, 2, 4, or 6; otherwise, a specification exception is recognized and a program interruption is caused.

Addition-Subtraction

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Multiplication-Division

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C6	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	F1

FIGURE 7. HEXADECIMAL ADDITION-SUBTRACTION AND MULTIPLICATION-DIVISION CHARTS

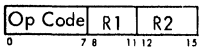


FIGURE 8. RR FORMAT

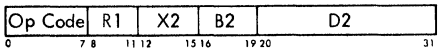


FIGURE 9. RX FORMAT

Data Formats

- Storage unit format.
- Working register format.

The floating-point data occupy a fixed-length double word format (Figure 10) or a single word format (Figure 11) in the main storage unit, and the double word format (Figure 12) or a single word format (Figure 13) in the floating-point registers. The floating-point registers are numbered 0, 2, 4, and 6.

The first bit in either the short or long main storage format is the sign bit (S). The next seven-bit positions are the characteristic (exponent), which is expressed in excess 64 with a range from -64 through +63. In the floating-point registers, the fraction is in bit positions 0-55 (long format) or 0-23 (short format); in both formats, the sign is bit 56 and the characteristic is in bit positions 57-63. In either format, a hexadecimal 00 represents the smallest value (-64), a hexadecimal 40 represents an exponent of 0, and a hexadecimal 7F represents the largest value (+63). See Figure 14. The remaining digits in either the short format (bits 8-31), or the long format (bits 8-63) of the main storage format are the number or fraction (mantissa) of the floating-point data word. Therefore, the fraction field is considered as either 6 or 14 hexadecimal digits in length, and the hexadecimal point is located to the left of the high-order digit of the fraction.

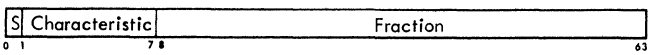


FIGURE 10. DOUBLE WORD FORMAT IN MAIN STORAGE

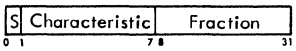


FIGURE 11. SINGLE WORD FORMAT IN MAIN STORAGE

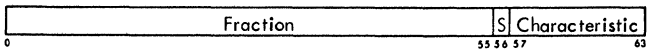


FIGURE 12. DOUBLE WORD FORMAT IN MAIN STORAGE

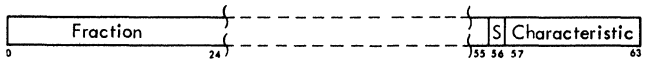


FIGURE 13. SINGLE WORD IN FLP REGISTER

Normalization

- A normalized number has the greatest precision.
- A normalized number has a nonzero high-order digit.
- Operations are performed with or without normalization.
- A zero characteristic, zero fraction, and plus sign is a true zero.

A quantity has the greatest precision when a floating-point number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit. If one or more high-order fraction digits (bits 8-11, 12-15, 16-19, etc.) are zero, the number is said to be unnormalized. The process of normalization consists of shifting the fraction to the left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A zero fraction cannot be normalized, and its associated characteristic remains unchanged when normalization is called for. Since normalization applies to hexadecimal digits, the three high-order bits of a normalized number may be zeros.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result; this function is called postnormalization. The operands are normalized prior to the arithmetic process in multiplication and division; this function is called prenormalization.

Most operations are performed in one or two ways; with or without normalization. However, addition and subtraction are specified either way. If an operation is performed without normalization, high-order zeros in the result fractions are not

Hexa-decimal	Decimal	Binary
00	-64	0000000
01	-63	0000001
02	-62	0000010
03	-61	0000011
04	-60	0000100
05	-59	0000101
06	-58	0000110
07	-57	0000111
08	-56	0001000
09	-55	0001001
0A	-54	0001010
0B	-53	0001011
0C	-52	0001100
0D	-51	0001101
0E	-50	0001110
0F	-49	0001111
10	-48	0010000
11	-47	0010001
12	-46	0010010
13	-45	0010011
14	-44	0010100
15	-43	0010101
16	-42	0010110
17	-41	0010111
18	-40	0011000
19	-39	0011001
1A	-38	0011010
1B	-37	0011011
1C	-36	0011100
1D	-35	0011101
1E	-34	0011110
1F	-33	0011111
20	-32	0100000
21	-31	0100001
22	-30	0100010
23	-29	0100011
24	-28	0100100
25	-27	0100101
26	-26	0100110
27	-25	0100111
28	-24	0101000
29	-23	0101001
2A	-22	0101010
2B	-21	0101011
2C	-20	0101100
2D	-19	0101101
2E	-18	0101110
2F	-17	0101111
30	-16	0110000
31	-15	0110001
32	-14	0110010
33	-13	0110011
34	-12	0110100
35	-11	0110101
36	-10	0110110
37	-09	0110111
38	-08	0111000
39	-07	0111001
3A	-06	0111010
3B	-05	0111011
3C	-04	0111100
3D	-03	0111101
3E	-02	0111110
3F	-01	0111111
40	00	1000000
41	+01	1000001
42	+02	1000010
43	+03	1000011
44	+04	1000100
45	+05	1000101
46	+06	1000110
47	+07	1000111
48	+08	1001000
49	+09	1001001
4A	+10	1001010
4B	+11	1001011
4C	+12	1001100
4D	+13	1001101
4E	+14	1001110
4F	+15	1001111
50	+16	1010000
51	+17	1010001
52	+18	1010010
53	+19	1010011
54	+20	1010100
55	+21	1010101
56	+22	1010110
57	+23	1010111
58	+24	1011000
59	+25	1011001
5A	+26	1011010
5B	+27	1011011
5C	+28	1011100
5D	+29	1011101
5E	+30	1011110
5F	+31	1011111
60	+32	1100000
61	+33	1100001
62	+34	1100010
63	+35	1100011
64	+36	1100100
65	+37	1100101
66	+38	1100110
67	+39	1100111
68	+40	1101000
69	+41	1101001
6A	+42	1101010
6B	+43	1101011
6C	+44	1101100
6D	+45	1101101
6E	+46	1101110
6F	+47	1101111
70	+48	1110000
71	+49	1110001
72	+50	1110010
73	+51	1110011
74	+52	1110100
75	+53	1110101
76	+54	1110110
77	+55	1110111
78	+56	1111000
79	+57	1111001
7A	+58	1111010
7B	+59	1111011
7C	+60	1111100
7D	+61	1111101
7E	+62	1111110
7F	+63	1111111

FIGURE 14. FLOATING-POINT EXPONENT VALUES

eliminated. The result may or may not be normalized, depending on the original operands.

In normalized and unnormalized operations, the initial operands need not be in normalized form. The intermediate fraction results are shifted right when an overflow occurs; the intermediate fraction result is truncated to the final result length after the shifting, if any.

A number with zero characteristic, zero fraction, and plus sign is called a true zero. As the result of an arithmetic operation, a true zero may arise because of the particular magnitude of the operands. A result is forced to be true zero when an exponent underflow occurs and the corresponding mask bit is off, or when a result fraction is zero and no program interruption due to significance exception is taken. When the program interruption is taken, the true zero is not forced, and the characteristic and sign of the result remain unchanged. When a divisor has a zero fraction, division is omitted, a floating-point divide exception exists, and a program interruption occurs. Otherwise, zero fractions and zero characteristics participate as normal numbers in all arithmetic operations.

The sign of a sum, difference, product, or quotient with zero fraction is positive. The sign of a zero fraction resulting from other operations is established by the rules of algebra from the operand signs.

Program Interrupts

- Exceptional instructions, data, or results cause program interrupts.

When an interrupt occurs, the current program status word (PSW) is stored. The interrupt code in the old program status word identifies the cause of the interrupt. The following exceptions cause a program interrupt in floating-point arithmetic:

Protection: The storage key of a result location does not match the protection key in the program

status word of a store instruction. The operation is suppressed; the condition code, data in the registers, and data in storage remain unchanged.

Addressing: An address designates a location outside the available storage for the installed system. The operation is terminated. The result data and the condition code are predictable, and no registers

Specification: A short operand is not located on a 32-bit boundary, a long operand is not located on a 64-bit boundary, a floating-point register address other than 0, 2, 4, or 6 is specified. The instruction is suppressed: the condition code, the data in the register, and the data in storage remain unchanged. The address restrictions do not apply to the components from which an address is generated (the content of the D2 field and the contents of the registers specified by X2 and B2).

Exponent Overflow: When exponent overflow occurs, the operation is completed and a program interrupt takes place. The fraction is normalized, and the sign and fraction of the result are correct. The result characteristic is 128_{10} smaller than the correct characteristic.

Floating-point add and subtract instructions set the condition code (PSW bits 34 and 35). If the result is zero, a 00 condition code is set. If the result is less than zero, a 01 condition code is set, and if the result is greater than zero, a 10 condition code is set.

Exponent Underflow: When exponent underflow occurs for a floating-point add, subtract, compare, multiply, or divide instruction, a program interrupt occurs if the exponent underflow mask bit is a one. The operation is completed, and the correct sign and normalized fraction are put away. The result exponent is 128_{10} greater than the correct exponent. When an exponent underflow occurs and the exponent mask bit is zero, the operation is completed by replacing the result with a true zero.

The condition code remains unchanged for halve, multiply, and divide instructions. For add and subtract instructions, the condition code is set to reflect the value of the result. If the result fraction is zero, a 00 condition code is set. If the result fraction is less than zero, a 01 condition code is set, and if the result is greater than zero, a 10 condition code is set.

Significance: The result fraction of an addition or subtraction is zero. A program interrupt occurs if the significance mask bit is a one. The mask bit also affects the result of the operation. When the significance mask bit is a one, the operation is completed without further change to the characteristic or the result. In either case, the condition code is set to 0.

Floating-Point Divide: Division by a number with zero fraction is attempted. The operation is suppressed; the condition code, data in the registers, and data in storage remain unchanged.

Condition Codes

- Sign-control, add, subtract, and compare instruction results set condition codes.
- Multiplication, division, load, and store instructions do not change condition codes.

The result of floating-point sign-control, add, subtract, and compare operations is used to set the condition code. Multiply, halve, divide, load, and store instructions leave the code unchanged. The condition code is used for decision-making by branch-on-condition instructions.

The condition code relfects two types of results for floating-point arithmetic. For most operations, the states 0, 1, or 2 indicate that the content of the result register is zero, less than zero, or greater than zero, respectively. A zero result is indicated whenever the result fraction is zero, including a forced zero. State 3 is never set by floating-point instructions.

For comparison, the states 0, 1, or 2 indicate that the first operand is equal, low or high. Figure 15 gives the condition code setting for floating-point arithmetic.

FLOATING-POINT INSTRUCTIONS

- Floating-point instructions have either long or short operands.
- Floating-point number consists of a signed exponent and fraction.
- The exponent is expressed in excess 64 binary notation.
- The fraction is expressed in hexadecimal.

Floating-point instructions have either long or short operands. Short-precision floating-point operands and results (except multiply) are 32-bit floating-point words; positions 24-55 of the floating-point register are not used or changed. The final result in short-precision instructions is six fraction digits; however, intermediate results in addition, subtraction, and division may be extended to a seventh digit.

Instructions	Condition Codes			
	0	1	2	3
Add (n)	RF is 0	RF < 0	RF > 0	Never Occurs
Add (u)	RF is 0	RF < 0	RF > 0	----------
Compare	Operands =	1st O is lo	1st O is hi	----------
Load and Test	RF is 0	RF < 0	RF > 0	----------
Load Complement	RF is 0	RF < 0	RF > 0	----------
Load Negative	RF is 0	RF < 0	----------	----------
Load Positive	RF is 0	----------	RF > 0	----------
Subtract (n)	RF is 0	RF < 0	RF > 0	----------
Subtract (u)	RF is 0	RF < 0	RF > 0	----------

RF = Result Fraction RE = Result Exponent 1st O = 1st Operand

oFIGURE 15. CONDITION CODE SETTING

This digit, called the guard digit, is to increase the precision of the final result. When long-precision floating-point is specified, the operand and result is a 64-bit floating-point word. The intermediate result in long precision may be extended to a fifteenth digit.

Floating-point multiply products always extend the entire length of the floating-point word (64 bits). In short precision, no significant digits are lost; however, in long precision, the low-order digits are lost due to shifting of the product beyond the capacity of the floating-point register.

A floating-point number consists of a signed exponent and a signed fraction. The exponent is expressed in excess -64 binary notation; the fraction is expressed as a hexadecimal number having a decimal point to the left of the high-order digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic, bits 1-7 of the floating-point format, indicates the power of the exponent.

The characteristic is treated as an excess -64 number with a range from -64 (binary value of 0000000) through +63 (binary value of 1111111). The range covered by the magnitude (M) of a normalized floating-point number is:

$$16^{-64} \leq M \leq 16^{63}$$

which is approximately

$$5.4 \times 10^{-78} \leq M \leq 7.2 \times 10^{75}$$

The floating-point arithmetic instructions and their mnemonic, format, and operation code are given in Figure 16. All operations are specified in short and long precision and are part of the floating-point feature. Figure 16 indicates when normalization occurs, when the condition code is set, and the exceptions that cause a program interrupt.

Add-Subtract

Addition of two floating-point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each hexadecimal digit of shift, until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit, and the characteristic is increased by one. If this increase causes a characteristic overflow, an exponent-overflow exception is signaled, and a program interruption occurs.

The short intermediate sum consists of seven hexadecimal digits and a possible carry. The long intermediate sum consists of 15 hexadecimal digits and a possible carry. The low-order digit is a guard digit retained from the fraction that is shifted right. Only one guard digit participates in the fraction addition. The guard digit is zero if no shift occurs.

After the addition, the intermediate sum 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 shift.

If normalization causes the characteristic to underflow and the underflow mask bit is zero, the characteristic and fraction are made zero. If normalization causes the characteristic to underflow and the corresponding mask bit is a one, the correct sign and fraction are put away. The exponent is 128_{10} greater than the correct exponent. If no left shift takes place, the intermediate sum is truncated to the proper fraction length, depending on the instruction being executed.

When the intermediate sum is zero and the significance mask bit is a one, a significance exception exists, and a program interruption takes

Name	Mnemonic	Type	Exceptions	Code
Add Normalized (Long)	NADR	RR	C,S,U,E,LS	2A
Add Normalized (Short)	NAER	RR	C,S,U,E,LS	3A
Add Normalized (Long)	NAD	RX	C,A,S,U,E,LS	6A
Add Normalized (Short)	NAE	RX	C,A,S,U,E,LS	7A
Add Unnormalized (Long)	AWR	RR	C,S,E,LS	2E
Add Unnormalized (Short)	AUR	RR	C,S,E,LS	3E
Add Unnormalized (Long)	AW	RX	C,A,S,E,LS	6E
Add Unnormalized (Short)	AU	RX	C,A,S,E,LS	7E
Subtract Norm (Long)	NSDR	RR	C,S,U,E,LS	2B
Subtract Norm (Short)	NSER	RR	C,S,U,E,LS	3B
Subtract Norm (Long)	NSD	RX	C,A,S,U,E,LS	6B
Subtract Norm (Short)	NSE	RX	C,A,S,U,E,LS	7B
Subtract Unnorm (Long)	SWB	RR	C,S,E,LS	2F
Subtract Unnorm (Short)	SUB	RR	C,S,E,LS	3F
Subtract Unnorm (Long)	SW	RX	C,A,S,E,LS	6F
Subtract Unnorm (Short)	SU	RX	C,A,S,E,LS	7F
Compare (Long)	CDR	RR	C,S	29
Compare (Short)	CER	RR	C,S	39
Compare (Long)	CD	RX	C,A,S	69
Compare (Short)	CE	RX	C,A,S	79
Divide (Long)	NDDR	RR	S,U,E,FK	2D
Divide (Short)	NDER	RR	S,U,E,FK	3D
Divide (Long)	NDD	RX	A,S,U,E,FK	6D
Divide (Short)	NDE	RX	A,S,U,E,FK	7D
Halve (Long)	HDR	RR	S,U	24
Halve (Short)	HER	RR	S,U	34
Load (Long)	LDR	RR	SS	28
Load (Short)	LER	RR	S	38
Load (Long)	LD	RX	A,S	68
Load (Short)	LE	RX	A,S	78
Load Positive (Long)	LPDR	RR	C,S	20
Load Positive (Short)	LPER	RR	C,S	30
Load Negative (Long)	LNDR	RR	C,S	21
Load Negative (Short)	LNER	RR	C,S	31
Load and Test (Long)	LTDR	RR	C,S	22
Load and Test (Short)	LTER	RR	C,S	32
Load Complement (Long)	LCDR	RR	C,S	23
Load Complement (Short)	LCER	RR	C,S	33
Multiply (Long)	NMDR	RR	S,U,E	2C
Multiply (Short)	NMER	RR	S,U,E	3C
Multiply (Long)	NMD	RX	A,S,U,E	6C
Multiply (Short)	NME	RX	A,S,U,E	7C
Store (Long)	STD	RX	P,A,S	60
Store (Short)	STE	RX	P,A,S	70

Notes:

A	Addressing exception	LS	Significance exception
C	Condition code is set	N	Normalized operation
E	Exponent-overflow exception	P	Protection exception
F	Floating-point feature	S	Specification exception
FK	Floating-point divide exception	U	Exponent-underflow exception

oFIGURE 16. FLOATING-POINT ARITHMETIC CODES

place. No normalization occurs; the intermediate sum characteristic remains unchanged. When the intermediate sum is zero and the significance mask bit is zero, the program interruption for the significance exception does not occur; rather, the characteristic is made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the result for floating-point add-subtract instructions is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

Compare

Comparison of two floating-point numbers consists of a characteristic comparison and a fraction subtraction. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each hexadecimal digit of shift, until the two characteristics agree. The fractions are subtracted algebraically; the sign, fraction, and exponent of each number are taken into consideration.

An exponent inequality is not decisive for magnitude determination since the fraction may have a different number of leading zeros. An equality is established by following the rules for normalized floating-point subtraction. When the intermediate sum, including a possible guard digit, is zero, the operands are equal. Neither operand is changed as a result of the operation, and exponent overflow, exponent underflow, or lost significance cannot occur.

Divide

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift may be called for. The intermediate quotient characteristic is adjusted for the shifts. All dividend fraction digits participate in forming the quotient, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to the desired number of digits. A program interruption for exponent overflow occurs when the final-quotient characteristic exceeds 127, and the operation is terminated. The correct sign and fraction are put away, and the characteristic is 128_{10} less than the correct characteristic.

A program interruption for exponent underflow occurs if the final-quotient characteristic is less than zero and the corresponding mask bit is a one. The sign and fraction are correct, and the characteristic is 128_{10} greater than the correct characteristic. If the corresponding mask bit is a zero, the result is made true zero and the interruption does not occur. Underflow is not signaled for the intermediate quotient or for the operand characteristics during prenormalization.

When division with a divisor with zero fraction is attempted, the 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 is zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interrupt for exponent underflow and exponent overflow. The program interrupt for significance is never taken for divide instructions.

Division is a non-restoring algorithm which incorporates a trial division by multiples and produces two quotient bits for each iteration cycle. A non-restoring approach is used because by following a trial subtraction which overdraws, with a trial addition, restoration cycles are eliminated.

The divisor is normalized by gating it through the main adder and shifter, and gating the result to the K register and the L register. The X3/2 divisor is generated by adding the contents of the K register to the contents of the L register shifted right 1 position. The result (X3/2 divisor) is placed in the L register.

The required divisor multiples are located in the M register and the L register. The X1 and X3/2 divisors are obtained by a direct readout from the registers, and the X1/2 and X3/4 divisors are obtained by shifting the registers right 1 position.

The quotient is assembled in the J register, and every second iteration, the J register is shifted left four bits by gating it to the register bus latch and back to the J register; thus, space is provided for the next four quotient bits.

The first step in each iteration cycle is the selection of the divisor. If the dividend is true, the decoding matrix shown in Figure 17 is used to select the multiple to be subtracted from the dividend; if the dividend is in complement, the matrix shown in Figure 18 is used. The quotient bits entered into the J register are shown in Figure 19. Most combinations produce two quotient bits; however, if the X3/4 divisor is used, three bits are generated. In this case, the third bit is retained and entered in place of the high-order quotient bit developed during the next iteration cycle.

The divide iteration cycles are continued until the shift counter content is reduced to one or three. At this time, the iteration cycles are terminated. If the last divisor used is the X3/4, the quotient is complete; however, if the X3/4 divisor is not the last divisor used, one more quotient bit is developed. The last quotient bit is generated by reducing the dividend by the X1 divisor.

Halve

The halve instructions divide the second operand by two by shifting the fraction right one bit. The result is placed in the first operand location. Normalization and test for zero fraction occurs.

Load

The load instructions transfer the second operand to the first operand location. The second operand is not changed, and neither exponent overflow, exponent underflow, nor lost significance can occur.

Load Type

The load type instructions transfer the second operand to the first operand location. The sign is made plus for the load positive (LPDR, LPER) instructions, or minus for the load negative (LNDR, LNER) instructions, or is changed to the opposite value for the load complement (LCDR, LCER) instructions, or the condition code is set for the load and test (LTDR, LTER) instructions.

True Dividend	Divisor			
	0.111	0.110	0.101	0.100
0.111	1	1	3/2	3/2
0.110	1	1	1	3/2
0.101	3/4	3/4	1	1
0.100	3/4	3/4	1	1
0.011	1/2	1/2	1/2	1/2
0.010	1/2	1/2	1/2	1/2
0.001	0	0	0	0
0.000	0	0	0	0

FIGURE 17. DIVISOR MULTIPLE SELECTION--TRUE DIVIDEND

Compl Dividend	Divisor			
	0.111	0.110	0.101	0.100
0.000	1	1	3/2	3/2
0.001	1	1	1	3/2
0.010	3/4	3/4	1	1
0.011	3/4	3/4	1	1
0.100	1/2	1/2	1/2	1/2
0.101	1/2	1/2	1/2	1/2
0.110	0	0	0	0
0.111	0	0	0	0

FIGURE 18. DIVISOR MULTIPLE SELECTION--COMPLEMENT DIVIDEND

Partial Dividend Result of Iteration		True		Compl	
		True	Compl	True	Compl
Multiple Used	3/2	11	10	01	00
	1	10	01	10	01
	3/4	011	010	101	100
	1/2	01	00	11	10
	0	00	--	--	11

FIGURE 19. QUOTIENT SELECTION DECODING

Multiply

Multiply consists of adding a multiple(s) specified by the decoding of the multiplier to the partial product. Multiplication is started at the low order end of the multiplier. The multiplier and partial product are shifted right four for each iteration.

In floating-point multiply, the digit normalized multiplicand is defined as the X16 multiple and is located in the M register. The X2, X4, and X8 multiples are obtained by shifting the M register right three, right two, or right one, respectively. The X6 multiple is obtained by shifting the X12 multiple (stored in the L register) right one; the X10 multiple is obtained by interrupting the normal iteration sequence to allow the X2 multiple and then the X8 multiple to be added to the partial product. The X14 multiple is obtained by a similar process using the X6 and then the X8 multiple. Therefore, the X2, X4, X6, X8, X10, X12, X14 and X16 multiples are obtained from two registers, and the only additional multiple to be concerned with is the X1 multiple. The only time the X1 multiple is needed is during the first iteration cycle if the low-order bit of the multiplier is a 1 bit.

The multiplier is decoded in groups of five bits. As an example, the multiplier $(105)_{10}$ is $(69)_{16}$. If this is implemented into a hexadecimal machine notation, the following configuration is realized:

$(105)_{10} = (69)_{16} = 0110\ 1001$

The first group of five bits is:

XXXX 0110 1001

first group

The low-order bit of the first group is always decoded as a zero; however, if the actual bit is a one, the X1 multiple is provided by gating the K register right four to the normal input of the main adder. In the example, the first group is decoded as:

0 1000

therefore, the X8 multiple is gated into the true/complement input of the main adder. A partial product consisting of the X8 multiple and the X1 multiple (gated right three) is obtained at the output of the main adder and is gated to the K register.

A different operation takes place if the high-order bit of the group is decoded as a one. If $(113)_{10}$ is equal to $(71)_{16}$, then the bit configuration is:

$(113)_{10} = (71)_{16} = 0111\ 0001$

and the first group:

1 0001

is decoded as:

1 0000

In the example, the high-order bit is a one and the low-order bit is decoded as a zero. The K register is gated right four into the normal input of the main adder and the X16 multiple is transferred in complement (-X16) to the true/complement input of the main adder. A partial product consisting of the -X16 and the X1 multiple is obtained at the output of the main adder. The output of the main adder is gated into the K register. The subtraction of the X16 multiple in the hexadecimal four-bit group is the same as subtracting the X1 multiple in the hexadecimal five-bit group. The first group of five bits is decoded and the multiples used are shown in the left-hand columns of Figure 5048, and the decoding of successive groups of five bits and the multiples used are shown in the right-hand columns of Figure 5048.

Two examples will help clarify how the floating-point multiply operation is performed. The first example is without a bit in the high-order position of the first group of five bits, and the second example contains a bit in the high-order position of the first group of five bits.

Example 1:

$(625)_{10}$ x $(105)_{10}$ = $(65, 625)_{10}$

$(271)_{16}$ x $(69)_{16}$ = $(10, 059)_{16}$

0010 0111 0001 x 0110 1001 = 0001 0000 0000 0101 1001

	0001	0011	1000	1000	X8 multiple
	0000	0010	0111	0001	X1 multiple
	0001	0101	1111	1001	partial product
	1110	1010	0110		X6 multiple (R4 shift plus R1 shift of X12 mult)
0001	0000	0000	0101	1001	product

Example 2:

$(625)_{10}$ x $(113)_{10}$ = $(70, 625)_{10}$

$(271)_{16}$ x $(71)_{16}$ = $(113E1)_{16}$

0010 0111 0001 x 0111 0001 = 0001 0001 0011 1110 0001

	0000	0010	0111	0001	0000	+X16 multiple
	1111	1101	1000	1110	1111	-X16 multiple (complemented)
	0000	0000	0010	0111	0001	X1 multiple
					1	Hot 1
	1111	1101	1011	0110	0001	partial product
	0001	0011	1000	1		X8 multiple (R4 shift for next iteration)
(carry)	0001	0001	0011	1110	0001	product

After any iteration cycle, the partial product is either correct or the X1 multiple less than the correct partial product with respect to the multiplier group for the next iteration. Therefore, in example 1, the first multiplier group is 0 1001 and the partial product is correct, but in example 2, the first multiplier group is 1 0001 and the partial product is the X1 multiple less than the partial product. Correction before overmultiplication is possible for all hexadecimal groups of the multiplier except the low-order group.

Figure 20 is a sample floating-point multiply problem, and is concerned only with the fraction. It is assumed that the exponent adder and its operation is fully understood; therefore, it is not mentioned further. The problem assumes a normalized multiplicand and consists of a five-digit register for simplicity.

In the problem, the multiplicand is $(625)_{10}$ or $(271)_{16}$ and the multiplier is $(404)_{10}$ or $(194)_{16}$. The product is $(252,500)_{10}$ or $(3DA54)_{16}$.

During the first floating-point cycle, the multiplicand is brought from the M register to the main adder. From the main adder, it is returned to the K register and the M register. The exponent (two low-order digits) is set to zero because the exponent is not transferred through the main adder. The multiplier is located in the J register by the time the first floating-point latch is set. For all practical purposes, the exponent is not shown. The exponent is set to zero during the first left-four shift to the register bus latch and the right-eight shift to the J register. Therefore, at the end of the cycle identified by the first floating-point latch (Figure 20), the J, K, and M registers contain the data shown.

During the time shown in the iteration preparation cycle (Figure 17), the X12 multiple is generated and placed in the L register. The contents of the K register and the complement of the M register shifted right two are added to obtain the X12 multiple which is placed in the L register. The J, K, and M register contents are not changed during this cycle.

The first multiplier group is decoded (-X12), and the X12 multiple is gated to the true/complement input of the main adder, complemented, added to zeros and returned to the K register. By the end of this cycle, the J register is shifted right four, the partial product is contained in the K register, and the L and M registers are unchanged.

During the second iteration trigger cycle, the -X6 multiple (L register right one to the true/complement input of the main adder) is added to the contents of the K register which is shifted right four. The result is returned to the K register. By the end of the second iteration cycle, the K register contains the new partial product, the L and M registers are unchanged and the next multiplier group is shifted into the low-order positions of the J register to decode the next multiple.

The third iteration trigger cycle gates the M register right three to the true/complement input of the main adder in true form and the K register is gated right four to the normal input of the main adder. The result is returned to the K register.

The next cycles put away the result fraction and the exponent in the register specified by the R1 field of the multiply instruction.

Store

The store instructions place one of the floating-point registers in core storage. The first operand (R1) is stored at the location specified by the second operand (X2 + B2 + D2). The first operand is not changed.

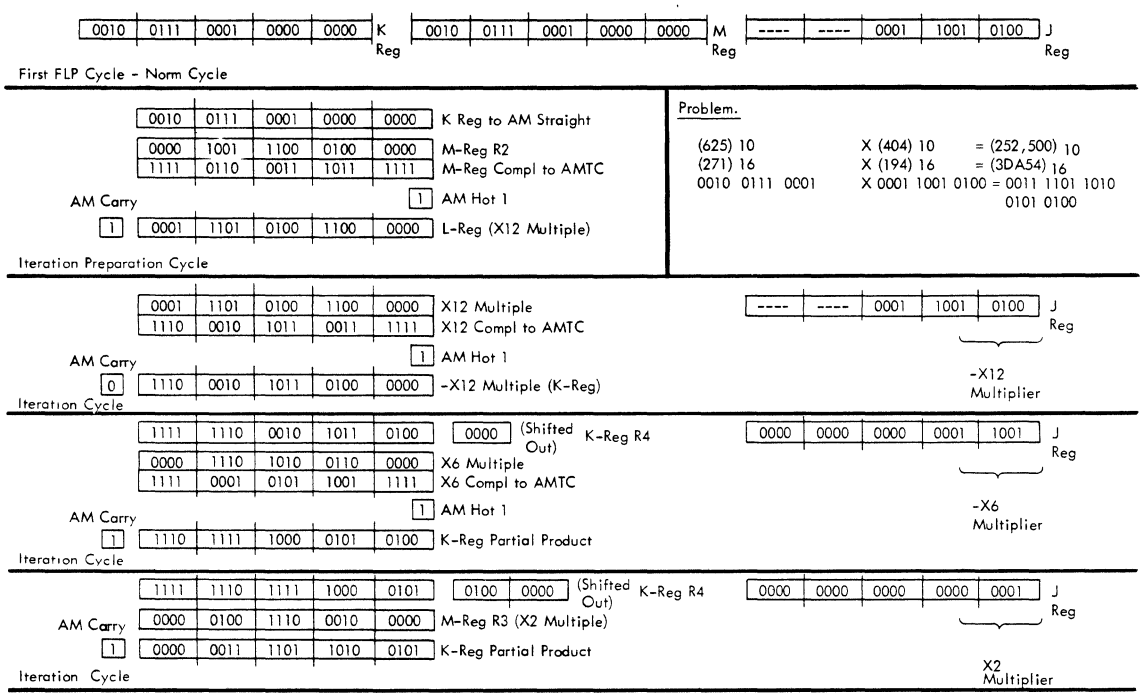


FIGURE 20. SIMPLE FLOATING-POINT MULTIPLY PROBLEM (FRACTION)

THEORY OF OPERATION

- Instructions are executed during intervals of time called cycles.
- The first machine cycle is the instruction cycle.
- The gating of operands continue after the I to E transfer.
- The sign triggers are set during T2 and the first FLP cycles.
- I time is followed by two or more execution cycles.

The IBM System/360 Model 75 instructions are performed during fixed intervals of time called cycles, and are identified by control triggers being turned on for one or more cycles. As an example, either an effective address is calculated during a fixed interval (cycle) of time identified by the T1 trigger during instruction time of each floating-point instruction, or data is transferred from register to register during a given cycle. The number of cycles necessary to execute a single instruction is dependent upon the instruction and other conditions within the machine. Examples of such conditions are: outstanding core storage requests, interrupts, and conditions blocking the turn-on of a control trigger.

Instructions consist of two or more parts, the operation code (op-code), and the operands specified by R1 and R2 or R1 and X2 + B2 + D2 fields. The operation code tells the machine what function it is to perform: add, subtract, compare, multiply, divide, store, etc. The operands are either addresses of floating-point registers (identified by the R1 and R2 fields of the RR instruction format and the R1 field of an RX instruction) or storage addresses (identified by the X2 + B2 + D2 fields of an RX instruction format).

The central processing unit operates in a prescribed sequence: the sequence is determined by the instruction being performed, and is performed during a fixed or data dependent (variable) interval of timed pulses.

The first machine cycle required to execute an instruction is called an instruction (I) cycle. It is assumed that the reader is familiar with the sequence of events taking place during the instruction cycle; therefore, the instruction cycle sequencing is only briefly reviewed here. Previous to the beginning of the instruction cycle:

1. The instruction is transferred from core storage to the AB register by an instruction request to core storage.

2. The instruction is transferred from the AB register to the instruction operation register (IOP reg) and the B operation register (BOP reg).

During instruction cycle time:

1. The operation is decoded in the IOP decoder and the BOP decoder.

2. The effective address (E) is calculated (R2 + D2 + X2).

3. The effective address is requested from core storage if the instruction is of the RX format or from the addressed floating-point register if the instruction is of the RR format.

4. A double word is requested from core storage to fill the A or B register if the present instruction emptied the register.

5. The gate select mechanism causes the length of the instruction (decoded from the first byte of IOP) to be added to the instruction counter in the incrementer.

6. This value is stored in the gate select register; at the I to E transfer this value is returned to the program status word (PSW) instruction counter, thus, updating the instruction counter.

7. The contents of the IOP register are transferred to the EOP register. This transfer is accomplished by one of the conditions shown in Figure 5550, depending on the last instruction type.

Floating-point instructions require operand to be gated after the I to E transfer. This gating is accomplished by the FLOUT control trigger (Figure 5057). FLOUT is set with the TN T2 trigger for any floating-point instruction regardless of the format. The RR format instructions allow two operands to be sequentially transferred from the floating-point registers to the working registers via the RBL. With TN T2 and ID RR FP, a control trigger, FR 2, is turned on (Figure 5057). This trigger causes the floating-point register, addressed by the R2 field, to be transferred during this and the following T2 cycles, if they occur. The I to E transfer removes the condition for turning on the FR 2 trigger; therefore, it turns itself off. When the FR 2 trigger is off, the first operand, addressed by the BR 1 field is selected for gating to the RBL by FLOUT.

The RX instructions allow one operand to be transferred from the floating-point register to the working registers via the RBL; the second operand (X2 + B2 + D2), which is calculated during the T1 cycle is requested from core storage and placed into the J register. FLOUT remains on until the E unit turns it off when gating is no longer required, normally during the first floating-point cycle. The exception is during prenormalization of the divisor; the divisor must be prenormalized before the second operand, the dividend, is accepted.

During the T2 cycle and the first floating-point cycle, the sign triggers (Figure 5551) are set. The R1 sign trigger contains the sign of the R1 operand and is loaded from bit 56 of the floating-point register addressed by the R1 field of either an RR or RX instruction. The R2 sign trigger contains the sign of the R2 operand and is loaded from bit 56 of the floating-point register addressed by the R2 field of a RR instruction, from J0 of an RX instruction with an even address, or from J32 of an RX instruction with an odd address. If during the T2 cycle, a RR instruction is decoded, the R2 sign trigger is set; if a RX instruction is decoded, the R1 sign trigger is set. Likewise, if during the first floating-point cycle, a RR instruction is decoded, the R1 sign trigger is set, or if a RX instruction is decoded, the R2 sign trigger is set.

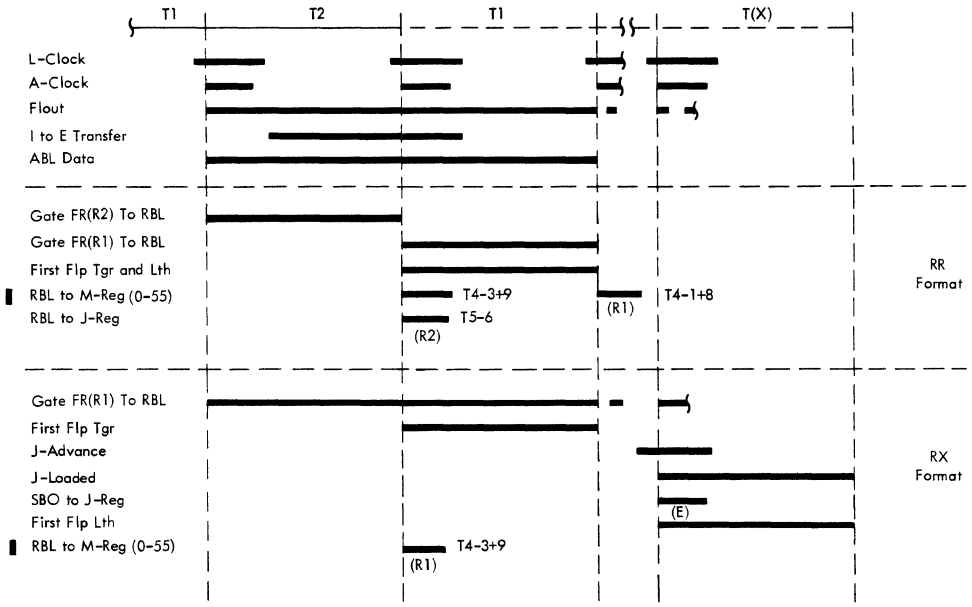
Figure 21 indicates the approximate timing of the I to E transfer, operand gating, and turn on of the first floating-point trigger and latch. Several possible conditions exist during T2 and to the I to E transfer.

First, in the RR format, the R2 operand is gated both to the J register and the M register. In Figure 5079, the gating to the J register is accomplished by the AND circuit at 6F, logic KU053, for the RR format instructions. The gating to the M register (Figure 5067) is accomplished by AND circuit at 6J, logic KU016, for a short floating-point RX format instruction. The register bus latch transfer to the J register is a right eight ring shift while the transfer from the register bus latch to the M register is a straight transfer. The right eight ring shift places the exponent located in bits 56-63 of the register bus latch in bits 0-7 of the J register. Also, note that the R1 operand is transferred from the register bus latch to the M register (Figure 5079) by AND circuit 6C, logic KU003 or AND circuit 6E, logic KU016, during the first floating-point cycle of compare, add, or subtract instructions; therefore, the M register contains the proper operand by the end of the first floating-point cycle.

Second, if the instruction is of the RX format, the R1 operand on the register bus latch is transferred to the M register by AND circuit 6J, logic KU003 or AND circuit 6G, logic KU016; the operand from core storage is loaded into the J register by the J advance pulse.

On compare, add, subtract and halve instructions, the transfer into M register bits 56-63 is blocked. This allows M register bits 56-59 to remain available to contain a possible guard digit. The operand 1 exponent is in the FP reg specified by BR1 of the instruction.

I time is followed by two or more cycles occurring during execution (E) time; the number of execution cycles required depends on the instruction being executed. Execution time begins as soon as the previous execution time is complete and the present instruction allows an I to E transfer to occur. The I to E transfer turns on the first floating-point trigger



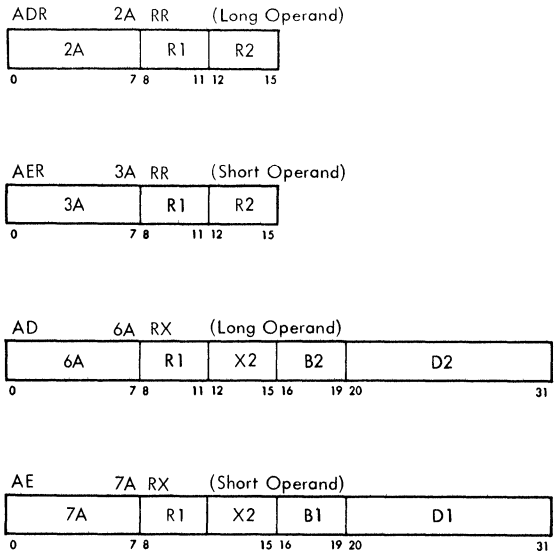
oFIGURE 21. FLP OPERAND TRANSFER TO WORKING REGISTERS

if the conditions shown in Figure 5552 are met.

When the execution time is started, the execution cycle is allowed to idle until data are received from core storage if the instruction is of an RX format. Part of the first execution cycle (first floating-point cycle) is performed until the effective operand data are received from core storage, indicated by the J loaded trigger (Figure 5552) being turned on. However, if data are requested from a floating-point register, such as during the RR instruction format, the execution time is allowed to proceed immediately after the I to E transfer by allowing the first floating-point latch (Figure 5552) to be set. When the R1 operand is located in the M register (RR or RX formats) and the data specified by the effective address (E) are located in the J register (RX format), the first floating-point latch is turned on and the first execution cycle (first floating-point cycle) is allowed to complete its operations. These operations are discussed in detail for each of the floating-point instructions in the following section of this manual. It is from this point (the first floating-point latch being turned on) that the following discussions for the floating-point instructions will begin.

ADD-SUBTRACT

The add normalized instruction formats are:



The add normalized instructions are handled as follows:

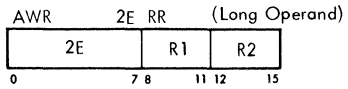
1. The characteristics of the two operands are compared.
2. The second operand is added to the first operand.
3. The result is normalized before it is placed in the first operand location.
4. Characteristic underflow when the corresponding mask bit is a zero causes a true zero condition.

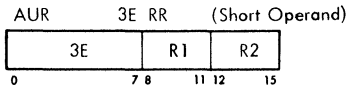
5. Characteristic underflow when corresponding mask bit is a one causes correct sign and normalized fraction to be put away. The result characteristic is 128_{10} greater than the correct characteristic.

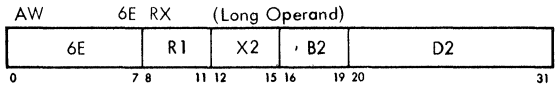
6. AER and AE instructions do not alter bits 24-55 of the floating-point register.

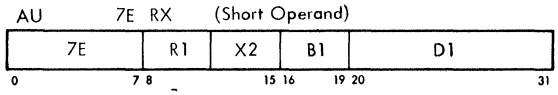
7. The sign of the result is derived by the rules of algebra.

The add unnormalized instructions formats are:





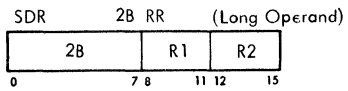


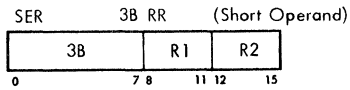


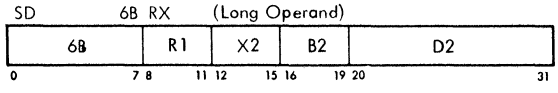
The add unnormalized instructions are handled as follows:

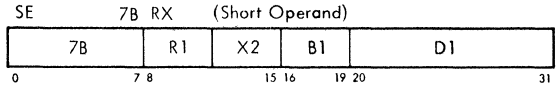
1. The characteristics of the two operands are compared.
2. The second operand is added to the first operand.
3. The result is placed in the first operand location.
4. ADR and AD instructions do not alter bits 24-55 fo the floating-point register.
5. The sign of the result is derived by the rules of algebra.

The subtract normalized instruction formats are:





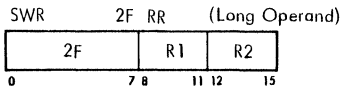


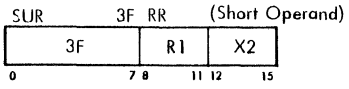


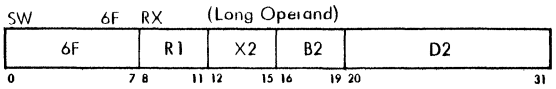
The subtract normalized instructions are handled as follows:

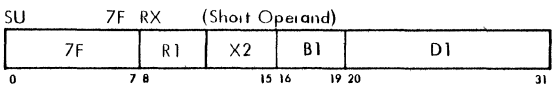
1. The characteristics of the two operands are compared.
2. The sign of the second operand is inverted before addition.
3. The second operand is subtracted from the first operand.
4. The result is normalized before it is placed in the first operand location.
5. SER and SE instructions do not alter bits 24-55 of the floating-point register.
6. The sign of the result is derived by the rules of algebra.

The subtract unnormalized instruction formats are:









The subtract unnormalized instructions are handled as follows:

1. The characteristics of the two operands are compared.
2. The sign of the second operand is inverted before addition.
3. The second operand is subtracted from the first operand.
4. SUR and SU instructions do not alter bits 24-55 of the floating-point register.
5. The sign of the result is derived by the rules of algebra.

Initial Operand Location

At the beginning of the execution cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single	FLP0-23, 56-63	J0-31
RR double	FLP0-63	J0-63
RX single (even address)	M0-23, FLP56-63	J0-31
RX single (odd address)	M0-23, FLP56-63	J32-63
RX double	M0-55, FLP56-63	J0-63

Instruction Sequencing

- Instruction sequencing is controlled by five triggers.
- Characteristic comparison is defined by the first floating-point cycle.
- Preshift is defined by the preshift trigger.
- Fraction addition is identified by the preshift-add trigger being on and the preshift trigger being off.
- Termination sequence is identified by the put-away and E last cycle triggers.
- Lost significance is detected during the first normalization cycle.
- Exponent overflow is caused when the exponent range is exceeded.
- Sign handling is performed during the adjustment cycle.

Figure 6400 is the data flow of the add and subtract instructions, and Figure 6401 is the logic flow of the add and subtract instructions.

First Floating-Point Cycle

The characteristic comparison is defined by the first floating-point latch (Figure 5552); it is a one cycle operation consisting of:

1. Obtaining the exponent difference.
2. Transferring the R2 or effective address operand from the J register to the K register with a left eight shift.
3. Transferring the R1 operand fraction from the floating-point register to the M register via the R RBL if the instruction is of the RR format.

The R2 or effective address exponent located in the J register bits 0-7 (RR formats, RX single even address, or RX double) or bits 32-39 (RX single odd address) is gated to the normal input of the exponent adder by the logic shown in Figure 5061. The R1 operand exponent located in the FLP register bits 56-63 is gated by the logic shown in Figure 5057 to the true/complement input of the exponent adder. The result is transferred from the AEOB to the exponent register and shift counter by the logic shown in Figures 5056 and 5086 respectively.

The R2 (or E) operand is gated from the J register to the true/complement input of the main adder. If the instruction is an RX and odd address format, the J register must be shifted left 32 positions in order to gate J32-63 into positions 0-31 of the main adder. This gating is accomplished by the logic shown in Figure 5061. If the instruction is an RR format, RX single even address format, or RX double format, the J register is gated straight to the main adder input without any shifting. All instructions shift the output of the main adder left eight bits (one byte) to eliminate the exponent (Figure 5087) and then set the AMOB output into the K register.

If the instruction is of the RR format, the R1 operand is gated from the floating-point register, specified by the R1 field of the instruction word, to the register bus latch. From the register bus latch, the R1 operand is gated to the M register as described earlier. The fraction is located in bits 0-23 for single precision RR formats, or in bits 0-55 for double precision RR formats. If the instruction is of the RX format, the R1 operand is already placed in the M register bits 0-23 for single precision RX format instructions or bits 0-55 for double precision RX formats; the exponent is in bits 56-63 of the floating-point register specified by BR1 of the instruction for all formats.

Preshift and Preshift-Add Cycles

- A variable cycle operation.
- Preshift identifies shifting of one operand until exponents are equal.
- Preshift-add identifies the add cycle.

The preshift and fraction addition cycles are a variable cycle operation identified by the preshift latch and the preshift-add latch (Figure 5553). If the difference between the two exponents resulted in the exponent adder halfsums for positions 1-7 all being equal to ones, the exponents are equal and preshifting of an operand is not required; therefore, the preshift trigger is not set because the line labeled -AE HS Eq 1 Lth line (Figure 5553) is active, thus, preventing the preshift trigger from being set.

A carry from the exponent adder high-order position (if the exponent adder HS is not equal to ones) indicates that the exponent of the K register (R2 or E operand) is larger than the exponent of the M register (R1 operand). No carry from the exponent adder high-order position indicates that the exponent of the

M register is larger than the exponent of the K register. An exponent difference greater than 64 sets the exponent overflow trigger because the capacity of the shift counter and exponent register is exceeded. The exponent overflow trigger signifies an exponent difference greater than 64, but it does not signify an interrupt condition.

Preshifting depends on an exponent difference and not on the magnitude of the difference. An exponent difference sets the preshift trigger (Figure 5553), which defines the preshift operation within the preshift-add sequence. A carry from the high-order position of the exponent adder gates the contents of the M register (R1 operand) to the main adder. The R1 operand is shifted right one or two hexadecimal digits (right four or right eight shift) depending on the shift counter value. The result is returned from the AMOB to the M register. No carry from the high-order position of the exponent adder gates the contents of the K register (R2 or E operand) to the main adder. The R2 or E operand is shifted right one or two hexadecimal digits depending on the shift counter value that is decoded in the shift counter decoder (Figure 5087). The result is returned to the K register from the AMOB latches. Whether a right four or right eight shift is taken during the first preshift cycle depends on the value of the shift counter bit 7. If the bit is a one, a right four shift is taken first, thus reducing the shift counter contents by one to an even amount. The following shifts, if any, are a right eight shift.

When either the K or M register is gated to the main adder and its contents are shifted, the shift counter is decremented by an amount equal to the number of hexadecimal digits (1 or 2) that the fraction is shifted. The exponent adder output is returned to the shift counter after each decrementing operation. When the shift counter is gated to the shift decoder during preshifting, the shift decoder determines the amount of the decrement and shift. The preshifting cycle is repeated until the shift decoder detects a shift count value equal to or less than two. When the shift counter value is equal to or less than two, the preshift cycles are terminated at the end of the current cycle because the exponents are equal.

The fraction addition cycle takes place if the characteristic comparison indicates the exponents are equal, or preshifting is completed indicating equal exponents.

The R1 operand in the M register is gated into the true complement input of the main adder and the R2 or E operand in the K register is gated into the normal input of the main adder. The result is returned to the K and M registers. The form of addition performed (true or complement) depends on the instruction and the operand signs. Figure 22 shows the form of addition performed.

Instruction	R1 Sign	R2 Sign	Operation Performed
Add	+	+	True
Add	+	-	Complement
Add	-	+	Complement
Add	+	-	True
Subtract	+	+	Complement
Subtract	+	-	True
Subtract	-	+	True
Subtract	-	-	Complement

FIGURE 22. ADD/SUBTRACT TRUE/COMPLEMENT ADDITION

The exponent of the fraction sum is the larger of the operand exponents. The R1 exponent is contained in the floating-point register; however, the R2 exponent has been lost. If the R2 exponent is the larger, it is equal to the R1 exponent plus the exponent difference obtained during the exponent comparison cycle.

During the fraction addition cycle, the exponent from the floating-point register specified by BR1 is gated to the exponent adder. If the R2 exponent is the larger, the exponent register that contains the exponent difference is gated to the normal input of the exponent adder. The exponent adder result is equal to the larger exponent and is returned to the shift counter and exponent registers. If the shift counter register is equal to or greater than 15, or if the exponent overflow trigger is on, it is detected by the SFT/DCR (Figure 5087) during the first preshift cycle, and the preshift trigger and exponent overflow trigger is turned off after one cycle. Exponent differences greater than 64 cannot be retained in the shift counter or exponent register because of the register length; therefore, a trigger must be set to indicate such conditions. The exponent overflow trigger is used to indicate this condition.

The register containing the fraction where the exponent is the larger is not preshifted. It is gated to the main adder, added algebraically to zero, and returned to the K and M registers. The fraction addition cycle completes the preshift-add sequence. The intermediate sum is contained in the K and M registers, and the exponent and sign of the intermediate sum is contained in the shift counter and the exponent registers.

PA and ELC Cycles

The PA trigger and ELC trigger define the termination sequence, Figures 5554 and 5555 respectively. It is a variable cycle operation and includes fraction recomplementation, normalization, exception handling, and result put-away.

A high-order carry from the main adder during a true add cycle indicates a fraction overflow. The intermediate sum, contained in the M register, is gated to the main adder, shifted right four bits, a one is forced into position 3, and the result is returned to the K register and the M register. The exponent register is gated to the exponent adder, incremented by one, and returned to the exponent register. The ELC trigger is set, and the resulting fraction and exponent is set into the floating-point register specified by R1 of the instruction format, and the instruction is terminated.

If a fraction overflow carry is not detected and the instruction is an add or subtract normalized instruction, an unnormalized sum is assumed and a normalization cycle is taken. The M register is gated to the main adder, complemented if the intermediate sum is in complemented form, and shifted by an amount depending on the number of high-order zeros. The result is returned to the K and M registers. The exponent register is gated to the exponent adder, decremented by an amount equal to the number of hexadecimal digits that the fraction is being shifted and returned to the exponent register. The decrement and shift amounts are determined by the SFT/DCR decoder (Figure 5087); normalization continues until the fraction is normalized or an exception condition is detected.

Add and subtract unnormalized instructions have a normalization cycle. The shift and decrement amounts are zero and the intermediate sum and exponent are not altered. The intermediate sum is examined for lost significance if the sum is not normalized. At the end of this cycle, either the significance adjustment or the put-away cycle follows.

Lost significance is detected during the first normalization cycle by the K register zero detector after recomplementing the intermediate sum, if it is required. Except for unnormalized add and subtract instructions, the entire K register is examined for lost significance. In short precision unnormalized add and subtract instructions, only the six high-order hexadecimal digits are examined, and in long precision unnormalized instructions, only the 14 high-order hexadecimal digits are examined because of the possibility of the significance digit (the guard digit) that is not part of the result fraction containing data.

If lost significance is detected, normalization is terminated and a significance adjustment cycle is taken. If the significance mask bit is a one, the exponent of the intermediate sum is the exponent of the result. The shift counter containing this exponent is gated to the exponent adder and returned to the exponent register. If the significance mask bit is a zero, the shift counter register is not gated to the exponent adder. The zero output of the exponent adder is returned to the exponent register.

The ELC trigger (Figure 5555) is set at this time or if the intermediate sum is normalized. The result exponent and fraction is set into the floating-point register specified by R1 of the instruction word. The instruction is terminated after one cycle of normalization if:

1. The intermediate sum is normalized, or
2. in the case of an unnormalized instruction with an unnormalized intermediate sum if lost significance is not detected.

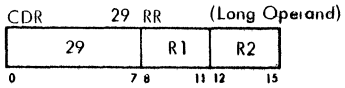
Exponent Overflow and Underflow: Exponent overflow is caused when the intermediate exponent is incremented beyond +64. The sequence is not altered by this occurrence, but the exponent overflow trigger is set.

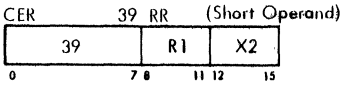
Exponent underflow may occur during any normalization cycle. The exponent underflow trigger is set, and, if the corresponding mask bit is a zero, normalization continues for one more cycle followed by an exponent underflow adjustment cycle. The ELC trigger is set, and the result exponent and fraction is set to zero and placed into the floating-point register specified by R1 of the instruction word, and the instruction is terminated. When the exponent underflow trigger is set and the corresponding mask bit is a one, normalization continues until the fraction is normalized. The ELC trigger is set, the result fraction and exponent are placed in the floating-point register specified by R1 of the instruction, and the instruction is terminated. The result exponent is 128_{10} greater than the correct exponent.

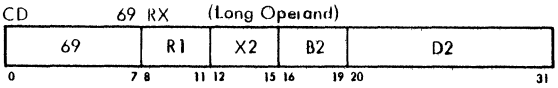
Sign Handling: Sign handling is performed during the adjustment cycle; the sign of the intermediate sum is determined by the sign of the R2 operand and the instruction performed. For add, the sign of the intermediate sum is set equal to the sign of the R2 operand. For subtract, the intermediate sum sign is set inverse to the sign of the R2 operand. If the intermediate sum is in true form, its sign is correct; if the intermediate sum is complement, the sign is inverted during the recomplement cycle. If lost significance or exponent underflow with the corresponding mask bit set to zero occurs, the sign of the result fraction is set to zero.

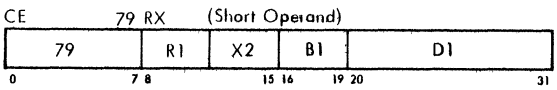
COMPARE

The compare instructions are:









The compare instructions are handled as follows:

1. The characteristic of the two operands is compared.
2. The sign of the second operand is inverted before addition.
3. The second operand is subtracted from the first operand.
4. The condition code indicates the result.
5. Short precision instructions do not check bits 24-55 of the FLP register.
6. Neither operand is changed as a result of the compare.
7. The comparison takes into consideration the sign, fraction, and exponent of each operand.

Initial Operand Location

At the beginning of the execution cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single	FLP0-23, 56-63	J0-31
RR double	FLP0-63	J0-63
RX single (even address)	M0-23, FLP56-63	J0-31
RX single (odd address)	M0-23, FLP56-63	J32-63
RX double	M0-55, FLP56-63	J0-63

Instruction Sequencing

- Instruction sequencing is controlled by five triggers.
- Characteristic comparison is defined by the first FLP trigger.
- Preshifting is defined by the preshift trigger.
- Fraction addition is defined by the preshift-add trigger.
- Termination sequence is identified by the PA and ELC triggers.

Figure 6402 is the data flow for the compare instructions, and Figure 6403 is the logic flow for the compare instructions.

First Floating-Point Cycle

The characteristic comparison is defined by the first floating-point latch (Figure 5552); it is a one cycle operation consisting of:

1. Obtaining the exponent difference.
2. Transferring the R2 or effective address operand from the J-register to the K register with a left eight shift.
3. Transferring the R1 operand from the floating-point register to the M register via the register bus latch if the instruction is of the RR format.

The R2 or effective address exponent located in J register bits 0-7, (RR formats, RX single even address, or RX double) or bits 32-39 (RX single odd address) is gated to the normal input of the exponent adder by the logic shown in Figure 5061. The R1 operand exponent, located in the floating-point register bits 56-63 is gated by the logic shown in Figure 5057, is gated to the true/complement input of the exponent adder. The result is transferred from the AEOB to the exponent register and shift computer by the logic shown in Figures 5056 and 5086 respectively.

The R2 or E operand is gated from the J register to the true/complement input of the main adder. If the instruction is an RX odd address format, the J register must be shifted left 32 positions in order to gate J32-63 into positions 0-31 of the main adder. This gating is accomplished by the logic shown in Figure 5061. If the instruction is an RR format, RX single even address format, or RX double format instruction, the J register is gated straight to the main adder input without any shifting. All instructions shift the output of the main adder left eight bits (one byte) to eliminate the exponent, (Figure 5087) and then set the AMOB output into the K register.

If the instruction is of the RR format, the R1 operand is gated from the floating-point register, specified by the R1 field of the instruction word, to the register bus latch. From the register bus latch, the R1 operand is gated into the M register as described previously. The fraction is located in bits 0-23 for single precision RR formats, or in bits 0-55 for double precision RR formats. If this instruction is of the RX format, the R1 operand is

already placed in M register bits 0-23 for single precision RX format instructions or bits 0-55 for double precision RX formats; the exponent is in bits 56-63 of the floating-point register specified by R1 of the instruction for all formats.

Preshift and Preshift-Add Cycles

- Variable cycle operations.
- Preshift identifies shifting of one operand until exponents are equal.
- Preshift-add identifies the add cycle.

The preshift and fraction addition cycles are variable cycle operations that are identified by the preshift latch and the preshift-add latch (Figure 5553). If the difference between the two exponents resulted in the exponent adder halfsums for positions 1-7 all being equal to ones, the exponents are equal and preshifting of an operand is not required; therefore, the preshift trigger is not set because the line labeled - AE HS Eq 1 Lth line (Figure 5553) is active, thus, preventing the preshift trigger from being set.

A carry from the exponent adder high-order position indicates that the exponent of the K register (R2 or E operand) is larger than the exponent of the M register (R1 operand). No carry from the exponent adder high-order position indicates that the exponent of the M register is larger than the exponent of the K register. An exponent difference greater than 64 sets the exponent overflow trigger because the capacity of the shift counter and exponent register is exceeded. The exponent overflow trigger signifies an exponent difference greater than 64, but it does not signify an interrupt condition.

Preshifting depends on an exponent difference and not on the magnitude of the difference. An exponent difference sets the preshift latch (Figure 5553), which defines the preshift operation within the preshift-add sequence. A carry from the high-order position of the exponent adder gates the contents of the M register (R1 operand) to the main adder. The R1 operand is shifted right one or two hexadecimal digits (right four or right eight shift) depending on the shift counter value. The result is returned from the AMOB to the M register. No carry from the high-order position of the exponent adder gates the contents of the K register (R2 or E operand) to the main adder. The R2 or E operand is shifted right one or two hexadecimal digits depending on the shift counter value that is decoded in the shift counter decoder (Figure 5087). The result is returned to the K register from the AMOB latches. Whether a right four or right eight shift is taken during the first preshift cycle depends on the value of the shift counter bit 7. If the bit is a one, a right four shift is taken first, thus reducing the shift counter contents by one to an even amount. The following shifts, if any, are a right eight shift.

When either the K or M register is gated to the main adder and its contents are shifted, the shift counter is decremented by an amount equal to the number of hexadecimal digits (1 or 2) that the fraction is shifted. The exponent adder output is returned to the shift counter after each decrementing operation. When the shift counter is gated to the shift decoder during preshifting, the shift decoder determines the amount of the decrement and shift. The preshifting cycle is repeated until the shift decoder detects a shift count value equal to or less than two. When the shift counter value is equal to or less than two, the preshift cycles are terminated at the end of the current cycle because the exponents are equal.

The fraction addition cycle takes place if either the characteristic comparison indicates the exponents are equal, or preshifting is completed indicating equal exponents.

The R1 operand in the M register is gated into the true complement input of the main adder and the R2 or E operand in the K register is gated into the normal input of the main adder. The result is returned to the K and M registers. The form of addition performed (true or complement) depends on the instruction operand signs. Figure 23 indicates the form of addition performed.

The exponent of the fraction sum is the larger of the operand exponents. The R1 exponent is contained in the floating-point register; however, the R2 exponent has been lost.

During the fraction addition cycle, the exponent from the floating-point register is gated to the exponent adder. If the R2 exponent is the larger, the exponent register, containing the exponent difference, is gated to the normal input of the exponent adder. The exponent adder result is equal to the larger exponent and is returned to the shift counter and exponent registers. If the shift counter register is equal to or greater than 15, or if the exponent overflow trigger is on, it is detected by the shift decoder (Figure 5087) during the first preshift cycle, and the preshift trigger and exponent overflow trigger are turned off after one cycle. Exponent difference greater than 64 cannot be retained in the

R1 Sign	R2 Sign	Operation Performed
+	+	Complement
+	-	True
-	+	True
-	-	Complement

FIGURE 23. TRUE/COMPLEMENT ADDITION

shift counter or exponent register because of the register length; therefore, a trigger must be set to indicate such conditions. The exponent overflow trigger is used to indicate this condition.

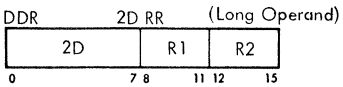
The register containing the larger exponent is not preshifted. It is gated to the main adder, added algebraically to zero, and returned to the K and M registers. The fraction addition cycle completes the preshift-add sequence. The intermediate sum is contained in the K and M registers, and the exponent and sign of the intermediate sum is contained in the shift counter and the exponent registers.

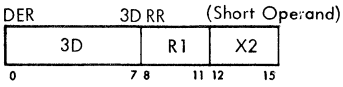
PA and ELC Cycles

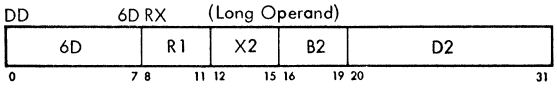
The termination sequence consists of result testing and is identified by the PA and ELC triggers being on. The result is tested, the condition code is set and the instruction is terminated. The condition code indicates the test result for the compare instructions. If the operands including the guard digit are equal, a condition code of 0 is set into the condition code register of the program status word; if the first operand is low, a condition code of 1 is set into the condition code register, and if the first operand is high, a condition code of 2 is set into the condition code register.

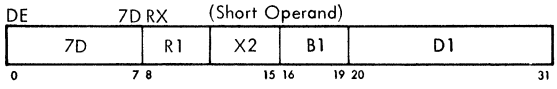
DIVIDE

The divide instructions are:









The divide instructions are handled as follows:

1. The first operand is divided by the second operand.
2. The quotient replaces the first operand.
3. The remainder is not retained.
4. DER and DE instructions do not alter bits 24-55 of the floating-point register.
5. Division consists of a characteristic subtraction and fraction division.
6. The difference between the exponents plus 64 is used as the quotient exponent.
7. The sign of the quotient is determined by the rules of algebra.

Initial Operand Location

At the beginning of the execution cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single	FLP register	J0-31
RR double	FLP register	J0-63
RX single (even)	FLP register	J0-31
RX single (odd)	FLP register	J32-63
RX double	FLP register	J0-63

Instruction Sequencing

- Instruction sequencing is controlled by 13 control triggers.
- The first floating-point latch identifies the prefetch cycle.
- The norm trigger identifies the divisor normalization cycles.
- The D2 trigger identifies the X3/2 divisor generation cycle.
- The D3 trigger identifies the dividend normalization cycles.
- The DL4 trigger identifies the cycle during which the dividend is made less than the divisor.
- Iteration preparation trigger identifies the first divide iteration cycle.
- The iteration trigger identifies the following divide iteration cycles.
- The first term trigger identifies the last quotient bit generation cycle.
- The quotient transfer/complement trigger identifies the quotient transfer cycle.
- The put-away and E last cycle triggers identify the quotient and exponent put-away cycle.

- The zero result trigger identifies a zero J register.

- The test trigger identifies a zero K register.

Figure 6404 is the data flow of the divide instructions, and Figure 6405 is the logic flow of the divide instructions.

First Floating-Point Cycle

The first floating-point latch identifies the first execution cycle for the divide instructions. During this cycle, the divisor that is located in the J register is gated through the main adder with a left eight shift to the K, L, and M registers. Figure 5087 shows the logic for the left eight shift controls. The divisor exponent (J0-7 RR single, RX even, RR double, RX double, or J32-39 RX odd) is gated from the J register through the exponent adder to the exponent register and the shift counter.

Owing to the nature of the operand fetch from the floating-point registers, the R1 sign trigger is set from the floating-point register bit 56 for the RR or RX instructions, and the R2 sign trigger is set from bit 0 of the J register for RR single, RR double, RX single even, and RX double, or from bit 32 of the J register for an RX single odd address instruction. Figure 5551 shows this logic gating to the R1 and R2 sign latches.

Norm Cycle

The norm trigger identifies the cycle(s) during which the divisor is normalized if necessary; the norm trigger (Figure 5556) is set following the first FLP cycle. The M register is gated to the main adder and the result is returned to the K register, L register, and the M register if M0-11 are zero. The gating of the M register to the main adder is shown in Figure 5067. The shift amount (left four or left eight) is decoded in the shift decoder (Figure 5087) and the shift amount (1 or 2) is subtracted from the contents of the shift counter

by gating the shift counter (Figure 5086) to the true/complement input of the exponent adder. The exponent adder output is returned to the exponent register and the shift counter. The K register is zero detected to determine if the divisor is zero. If the devisor is zero, the block put away trigger, E interrupt trigger, and test trigger are set after the first normalization cycle. When M0-11 are not zero, the output from the main adder is returned to the K register and the L register; the dividend is gated from the floating-point register through the RBL to the J register and the M register by the logic shown in Figure 5079. The entire contents of the register bus latch is gated to the J register. If the divide instruction is a long operand instruction, the entire contents of the register bus latch is also gated to the M register; however, if the divide instruction is a short operand instruction, only bits 0-23 and 56-63 of the register bus latch are gated to the M register.

D2 Cycle

When bits 0-11 or the M register are other than zero, the D2 trigger is set to identify the multiple generation cycle (Figure 5557). During this cycle, the X3/2 multiple is generated by gating the K register to the normal input of the main adder and the L register right one to the true/complement input of the main adder. The result is placed in the L register. The normalized divisor exponent is gated from the exponent register to the normal input of the exponent adder, and the dividend exponent is gated from bits 56-63 of the M register to the true/complement input of the exponent adder. The difference is placed in the exponent register. At this time, the J register is zero detected and if the contents of the J register are zero, the zero result trigger is set during the next cycle and the K register and the exponent registers are set to zero.

During this cycle, positions 0-7 of the K register are gated via the left byte gate to the digit buffer and digit counter register. The output of the digit buffer and digit counter are used to decode the number of high-order zeros contained in the divisor after it is digit normalized but not bit normalized.

D3 Cycle

If the dividend (J register) is not zero, the D3 trigger is set (Figure 5558) and the dividend is normalized during this and following D3 cycles. The divisor (K register) is gated to the register bus latch and the dividend (M register) is gated to the true/complement input of the main adder. The output of the main adder is returned to the M register until M0-3 are not zeros. Each time the M register is gated to the main adder, the exponent register is gated to the exponent adder, decremented by one for each digit that the dividend is normalized and the result is returned to the exponent register. When M0-3 are other than zero, the M register is again gated to the main adder; however, this time the AMOB is returned to the K register and the contents of the register bus latch (divisor) are gated to the M register.

During the last D3 cycle, the divisor multiple is decoded, the shift counter is set to 12 for single word iteration count or to 28 for double word iteration count, and the AEOB is returned to the exponent register. At the end of the last D3 cycle, the K register contains the dividend, the M register contains the divisor, the L register contains the X3/2 divisor, the exponent register contains the exponent difference, and the shift counter contains the iteration count.

If the dividend is decoded as being larger than the divisor during the D3 cycle(s), the DL4 trigger is set during the next cycle: however, if the dividend is not decoded as being larger than the divisor by the divide decoder during the D3 cycle(s), the iteration preparation trigger is set during the next cycle.

DL4 Cycle

If the dividend is decoded as being equal to or larger than the divisor during the D3 cycle, the DL4 trigger (Figure 5559) is set and the dividend is made smaller than the divisor. During the DL4 cycle, the dividend (K register) is gated to the normal input of the main adder, shifted right four, and the AMOB is returned to the K register. The exponent register (quotient exponent) is gated to the true/complement input of the exponent adder and a one is added to the quotient exponent. The result (exponent +1) is returned to the exponent register.

When the DL4 cycle is taken, it signifies that the dividend is larger than the divisor and the first divisor multiple must be decoded again. The new divisor multiple is determined by decoding the six high-order bits of the divisor that are contained in the digit buffer and digit counter, and the contents of bits 0 and 1 of the K register prior to the right shift. After the shift cycle, bits 0 and 1 are located in bits 4 and 5 of the K register as a result of the right four shift. At the end of the DL4 cycle the iteration preparation trigger is set and the first divide iteration occurs.

Iteration Preparation Cycle

The first divide iteration cycle is identified by the iteration preparation trigger (Figure 5560) being

turned on either after the D3 trigger if the divisor is larger than the dividend or after the DL4 trigger if the dividend was larger than the divisor. The K register is gated to the normal input of the main adder and the divisor multiple is gated to the true/complement input. The two inputs to the main adder are either added or subtracted, and the result is returned to the K register. At the same time, the quotient bits are inserted into J59 and J60 by the quotient insert logic (Figure 5061), while zeros are being read into the J-register from the RBL. At the same time the divisor multiple is gated to the true/complement input of the main adder, the shift counter is gated to the exponent adder true/complement input and one is subtracted from the value of the shift counter. The result (SC-1) is returned to the shift counter, the next divisor multiple is decoded from the contents of the digit buffer and digit counter and the output of bits 2-7 of the main adder. If a shift overflow or a quotient overflow occurs, a divide check will not result as is the case in fixed-point divide. However, if a quotient overflow occurs during the first iteration, two less iterations are taken and the first term trigger is set when the shift counter equals three rather than when it equals one.

Iteration Cycle

The following divide iteration cycles (2-10, 2-12, 2-26, or 2-28) are identified by the iteration trigger, (Figure 5561) being set. The K register is gated left two to the normal input of the main adder and either the M register or the L register is gated straight or right one to the true/complement input of the main adder. If the result of the previous iteration cycle is in complement form, the divisor multiple is gated to the true input of the true/complement input of the main adder. If the previous iteration cycle is in true form, the divisor multiple is gated to the complement input of the true/complement input of the main adder. The result is returned to the K register, and bits 2-7 of the adder sum are used to determine the next divisor multiple when the carry is received from the remaining halfsum bits. The quotient bits for the second iteration (SC shift counter is odd) are gated into J61 and J62 (Figure 5061).

The contents of the J register (quotient) are gated left four to the RBL each iteration cycle, but they are gated back into the J register only when the shift counter is even. Therefore, during iteration number one (identified by the iteration preparation trigger), the contents are not gated even though the shift counter is even, but the zero content of the register bus latch is gated into the J register and bits 59 and 60 are set with the first two quotient bits. On the second iteration cycle (identified by the iteration trigger being on) the shift counter is odd; therefore, the quotient bits are inserted into J61 and J62. The J register is read out left four to the RBL, but the RBL is not gated back to the J register; therefore, the J register contents are not lost or changed (except J61 and J62) because the J register is not released (reset); when the shift counter is odd.

The shift counter is decremented by one during each iteration cycle by gating it to the exponent adder and subtracting one from it. The result is returned to the shift counter. After the second iteration cycle, the shift counter is even and the K register and divisor multiple are added or subtracted in the normal manner. The J register is gated left four to the register bus latch and the register bus latch is returned to the J register while the quotient bits for the third iteration cycle are gated into J59 and J60. The shift counter is decremented by one and this process continues until the shift counter equals either one or three. When the shift counter equals either one or three, the iteration cycles are terminated. Whether the iteration sequence is terminated when the shift counter equals one or three depends on a quotient overflow being detected during the first iteration cycle. If a quotient overflow is detected, the iteration sequence is terminated when the shift counter equals three.

First Term Cycle

The first term cycle is identified by the first term trigger (Figure 5562) and is used to generate the last quotient bit. The K register is gated left two to the normal input of the main adder and the selected multiple is gated to the true/complement input of the main adder. The result is returned to the K register and the quotient insert logic (Figure 5061) inserts the last quotient bit into bit 63 of the J register.

Quotient Transfer/Complement Cycle

The quotient transfer cycle is identified by the quotient transfer/complement trigger (Figure 5563) being set. The quotient fraction (J register) is gated through the main adder with a left eight shift to the K register. The exponent register containing the quotient exponent is gated to the exponent adder; if the quotient overflow trigger is on, one is added to the quotient exponent. The exponent adder bit zero (fraction sign bit) is set plus or minus according to the rules of algebra. The result is returned to the exponent register.

Zero Result Cycle

If during the time the D2 trigger is set, the dividend (J register) is detected as all zeros, the zero result

trigger (Figure 5564) is set during the next cycle instead of the D3 trigger. The K register and the exponent register are set to zero by gating the AMOB and the AEOB to them, respectively. The E last cycle is the next cycle following the zero result cycle.

Test Cycle

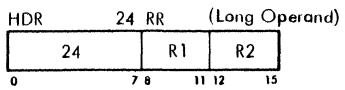
The test cycle is identified by the test trigger (Figure 5565) being turned on if the exponent underflow trigger is on and the underflow mask bit is a zero. The test trigger is turned on after the put-away cycle or if the K register (divisor) is detected as all zeros during the time the norm trigger is on. The K register and the exponent registers are set to zero by gating the AMOB and the AEOB to them, respectively. The E last cycle is the next cycle following the test cycle.

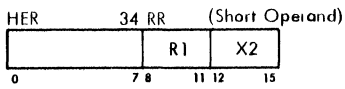
PA and ELC Cycles

The put-away cycle is identified by the PA trigger (Figure 5554) and the E last cycle trigger (Figure 5555) being set. If an exponent underflow did not occur or if the underflow mask bit is a one, the ELC trigger is set with the put-away trigger during this cycle. During this cycle, the quotient fraction located in the K register and the quotient exponent located in the exponent register are transferred to the floating-point register specified by the ER1 register. If the exponent underflow trigger is on and the underflow mask bit is a zero, the test trigger is set during the next cycle, but the E last cycle trigger is not set.

HALVE

The halve instructions are:





The halve instructions are handled as follows:

1. The second operand is divided by shifting the fraction right one bit.
2. The normalized quotient is placed in the first operand location.
3. The HER instruction does not alter bits 24-55 of the floating-point register.
4. Normalization and test for zero fraction occur. Lost significance interrupt does not occur. Lost significances cause a true zero result.

Initial Operand Location

At the beginning of the exeuction cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single	None	M0-23, J0-7
RR double	None	M0-55, J0-7

Instruction Sequencing

- Instruction sequencing is controlled by three triggers.
- Halve cycle is identified by first floating-point trigger.
- Termination cycle is identified by the E last cycle trigger.

Figure 6406 is the data flow of the halve instructions, and Figure 6407 is the logic flow of the halve instructions.

First Floating-Point Cycle

The halve cycle is a one-cycle operation identified by the first floating-point trigger (Figure 5552) being set. The operation consists of gating the M register (Figure 5067) right one to the main adder true/complement input, setting the output of the main adder into the K register and M register, gating the R2 exponent from the J register (bits 0-7) to the normal input of the exponent adder, and gating the exponent adder output into the exponent register.

PA and ELC Cycles

The PA and ELC triggers define the termination sequence (Figures 5554 and 5555), a variable cycle operation that includes normalization, exception handling and result put-away.

An unnormalized sum is assumed and a normalization cycle is taken. The M register is gated to the main adder and shifted by an amount depending on the number of high-order zeros. The result is returned to the K register and M register. The exponent register is gated to the exponent adder, decremented by an amount equal to the number of hex digits that the fraction is being shifted and returned

to the exponent register. If the sum is already normalized, the shift and decrement amounts are zero, and the intermediate sum and exponent are not altered. The decrement and shift amounts are determined by the SFT/DCR decoder (Figure 5087); normalization continues until the fraction is normalized or an exception condition is detected.

Lost significance is detected during the first normalization cycle by the K register zero detector. The entire K register is examined for lost significance. If lost significance is detected, normalization is terminated and a significance adjustment cycle is taken. The shift counter register is not gated to the exponent adder, and the zero output of the exponent adder is returned to the exponent register. Lost significance does not cause a program interrupt regardless of the setting of the significance mask bit.

The ELC trigger (Figure 5555) is set at this time or if the intermediate sum is normalized. The result fraction and exponent are set into the floating-point register specified by R1 of the instruction word.

Exponent Underflow

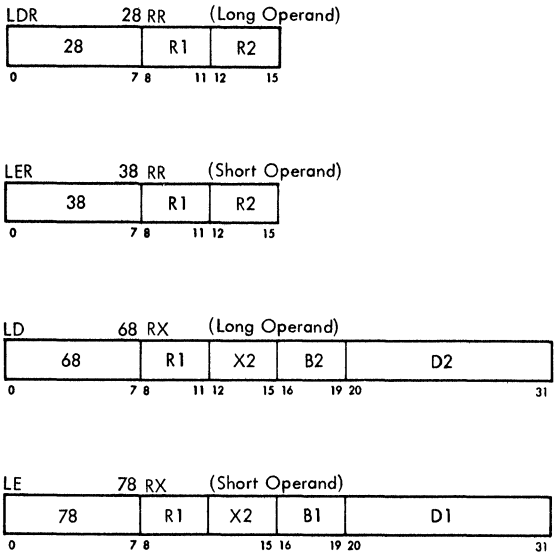
Exponent underflow may occur during any normalization cycle. The exponent underflow trigger is set and if the exponent underflow mask bit is a zero, normalization continues for one more cycle and an exponent underflow adjustment cycle is taken. The ELC trigger is set, the result fraction and exponent are set to zero and placed in the floating-point register specified by R1 of the instruction word, and the instruction is terminated. When the exponent underflow trigger is set and the underflow mask bit is a one, normalization continues until the fraction is normalized. The ELC trigger is set, the result fraction and exponent are placed in the floating-point register specified by R1 of the instruction word, and the instruction is terminated.

Sign Handling

Sign handling is performed during the adjustment cycle; the sign of the intermediate sum is determined by the sign of the R2 operand. If lost significance or an exponent underflow occur, with the underflow mask bit set to a zero, the sign of the result fraction is set to zero.

LOAD

The load instructions are:



The load instructions are handled as follows:

1. The second operand is placed in the first operand location.
2. The second operand is not changed.
3. The LE and LER instructions do not alter bits 24-55 of the floating-point register.
4. Exponent overflow, underflow, or lost significance cannot occur.

Initial Operand Location

At the beginning of the execution cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single	None	J0-31
RR double	None	J0-63
RX single (even address)	None	J0-31
RX single (odd address)	None	J32-63
RX double	None	J0-63

Instruction Sequencing

- Instruction sequencing is controlled by two triggers.
- The transfer of the second operand is identified by the first floating-point trigger.
- Termination of the load instruction is identified by the E last cycle trigger.

Figure 6408 is the data flow of the load instruction, and Figure 6409 is the logic flow of the load instruction.

First Floating-Point Cycle

The second operand transfer is a one-cycle operation identified by the first floating-point trigger (Figure 5552) being set. During this cycle, the second operand (RX or X2 + B2 + D2) fraction is transferred from the J register (J0-31 RR single, RX single even address, J32-63 RX single odd address, or J0-63 RR double and RX double) to the true/complement input of the main adder (Figure 5061). The output of the main adder is shifted left 8 positions (Figure 5087) and the result is placed in the K register.

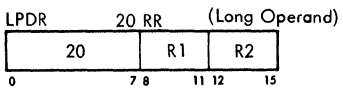
During the transfer of the J register to the main adder, the exponent (J0-7 RR single, RR double, RX single even address, and RX double, or J32-39 RX single odd address) is transferred to the normal input of the exponent adder (Figure 5061). The output of the exponent adder is placed in the exponent register (Figure 5056).

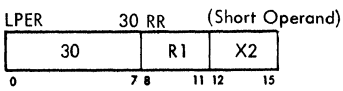
ELC Cycle

The one cycle put-away sequence is identified by the E last cycle trigger (Figure 5555) being set at the end of the second operand transfer cycle. The fraction is transferred from the K register to the floating-point register identified by R1 of the floating-point load instruction. The exponent is transferred from the exponent register to bits 56-63 of the floating-point register. When the fraction and exponent are located in the floating-point register, the instruction is terminated.

LOAD TYPE

The load positive type instructions are:



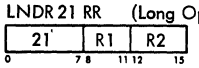


The load positive type instructions are handled as follows:

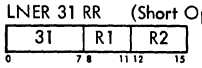
1. The second operand is placed in the first operand location.
2. The sign of the second operand is made plus.
3. The characteristic and fraction are not changed.
4. The LPER instruction does not alter bits 24-55 of the floating-point register.

The load negative type instructions are:

LNDR 21 RR (Long Operand)



LNER 31 RR (Short Operand)

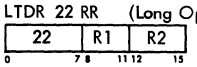


The load negative type instructions are handled as follows:

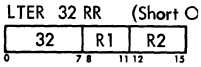
1. The second operand is placed in the first operand location.
2. The sign of the second operand is made minus.
3. The characteristic and fraction are not changed.
4. The LNER instruction does not alter bits 24-55 of the floating-point register.

The load and test type instructions are:

LTDR 22 RR (Long Operand)



LTER 32 RR (Short Operand)

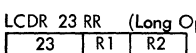


The load and test type instructions are handled as follows:

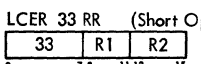
1. The second operand is placed in the first operand location.
2. Its sign and magnitude determine the condition code.
3. The second operand is not changed.
4. The LTER instruction does not alter bits 24-55 of the floating-point register.
5. The LTER instruction does not test bits 24-55 of the floating-point register.
6. When the first and second operand are the same register, the operation is equivalent to a test without data movement.

The load complement type instructions are:

LCDR 23 RR (Long Operand)



LCER 33 RR (Short Operand)



The load complement type instruction are handled as follows:

1. The second operand is placed in the first operand location.
2. The sign is changed to the opposite value.
3. The characteristic and fraction are not altered.
4. The LCER instruction does not alter bits 24-55 of the floating-point register.

Initial Operand Location

At the beginning of the execution cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single	None	J0-31
RR double	None	J0-63

Instruction Sequencing

- Instruction sequencing is controlled by two triggers.
- The second operand transfer is identified by the first floating-point trigger.
- The termination cycle is identified by the E last cycle trigger.

Figure 6410 is the data flow of the load type instructions, and Figure 6411 is the logic flow of the load type instructions.

First Floating-Point Cycle

The second operand transfer is a one-cycle operation identified by the first FLP trigger (Figure 5552) being set. The one-cycle operation consists of gating the second operand from the J register (J0-31 RR single, RX single even address, or J32-63 RX single odd address, or J0-63 RR double, RX double) to the true/complement input of the main adder (Figure 5061). The output of the main adder is shifted left 8 positions (Figure 5087) and the result is placed in the K register.

During the transfer of the J register to the main adder, the exponent (J0-7 RR single, RR double, RX single even address, or RX double, or J32-39 RX single odd address) is gated to the normal input of the exponent adder (Figure 5061). If the instruction is an LPDR, LPER, LNDR, LNER, LCDR, or LCER instruction, the sign is set to the desired

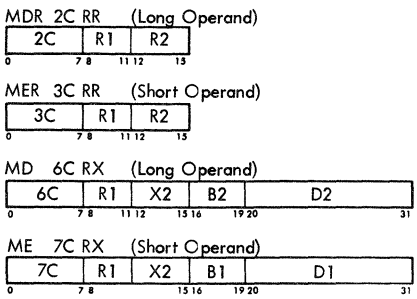
value during this cycle. The output of the exponent adder is placed in the exponent register. The load and test instructions (LTDR and LTER) do not alter the sign of the second operand on its transfer.

ELC Cycle

The load type instruction termination is a one-cycle put-away cycle identified by the E last cycle trigger (Figure 5555) being turned on at the completion of the transfer cycle that is identified by the first floating-point trigger. The result fraction is gated from the K register to the floating-point register specified by R1 of the load type instruction, and the exponent is gated from the exponent register to bits 56-63 of the floating-point register. If the instruction is the load and test type (LTDR or LTER), the sign and magnitude of the second operand determine the condition code setting of bits 34 and 35 of the program status word. After the exponent and fraction are set into the floating-point register and the condition code is determined (LTDR and LTER instructions), the load type instruction is terminated.

MULTIPLY

The multiply instructions are:



The multiply instructions are handled as follows:

1. The multiplicand (M register) is normalized by gating it to the true/complement input of the main adder and shifting the output 0, 1, or 2 hexadecimal digits.
2. Bits 56-63 of the M register are gated to the true/complement input of the exponent adder; the result (minus shift amount) is placed in the exponent register.
3. The multiplier is transferred from the RBL to the J register with a right eight ring shift.
4. The K register and the M register are gated to the main adder to generate the X12 multiple.
5. The J register exponent and exponent register are added to obtain the product exponent.
6. Bits 59-63 of the J register are decoded to determine the multiple(s) used for each iteration cycle.
7. The multiplier and partial product are shifted four bits for each iteration.
8. The shift counter determines the number of iterations taken.
9. The normalized product and exponent replaces the first operand.
10. The ME and MER instructions do not alter bits 24-55 of the first operand.
11. A final product exponent overflow causes a program interrupt.
12. An overflow exception does not occur for an intermediate product exponent when the final exponent is brought within range by normalization.
13. An all zero product fraction sets the product sign and characteristic to zero; a program interrupt does not occur.
14. The program interrupt for lost significance is not taken for multiply instructions.

Initial Operand Location

At the beginning of the execution cycle, the operands are located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand	R2 Operand
RR single (MER)	FLP register	M0-23, 56-63 *
RR double (MDR)	FLP register	M0-63 *
RX single (ME) even address	M0-23, 56-63 *	J0-63
RX single (ME) odd address	M32-63 *	J0-63
RX double (MD)	M0-63 *	J0-63

* denotes multiplicand

Instruction Sequencing

- Instruction sequencing is controlled by eight triggers.
- Prenormalization is identified by the first floating-point trigger and the norm trigger.
- The X12 multiple generation is identified by the iteration preparation trigger.
- The multiple decode cycles are identified by the iteration preparation trigger and the iteration trigger.
- A second cycle during an iteration cycle is identified by the add trigger.
- A zero result is identified by the test trigger.

- The product and exponent of the multiply operation is gated to the floating-point register, which is identified by the put away trigger.

- The termination cycle is identified by E last cycle trigger.

Figure 6412 is the data flow of the multiply instructions, and Figure 6413 is the logic flow of the multiply instructions.

First Floating-Point Cycle

The first multiplicand pre-normalization cycle is identified by the first FLP trigger (Figure 5552) being set. The multiplicand (M register) (Figure 5067), is transferred through the main adder left shifted 0, 4, or 8 depending on the number of high-order zeros detected by the shift decoder (Figure 5087) and the result is placed in the K and M registers. At the same time that the M register is transferred to the main adder, bits 56-63 of the M register are gated to the exponent adder and 0, 1, or 2 is subtracted from the exponent depending on the amount of shift (0, left four, or left eight) that is decoded to eliminate high-order (leading) zero digits in the multiplicand. The result from the exponent adder is placed in the exponent register. During this cycle, the multiplier is transferred from the register bus latch to the J register (RR formats) with a right eight ring shift to locate the exponent in J0-7 by the logic shown for the right eight ring shift on Figure 5061.

Norm Cycle

The following multiplicand pre-normalization cycle(s) is identified by the norm trigger (Figure 5556) being set. The multiplicand (M register) is transferred to the main adder and the exponent register is transferred to the exponent adder; an appropriate shift and exponent decrement cycle is taken. The norm cycle(s) continues until bits 0-3 of the M register are hexnormalized.

The K register is used to zero detect the multiplicand. If the K register is not zero, pre-normalization cycle(s) is taken until the multiplicand is normalized; however, if the multiplicand fraction is zero, the test cycle trigger is turned on for the following cycle (third cycle). The test cycle zeros the exponent register and turns on the E last cycle trigger, which allows a true zero to be gated to the floating-point register.

Iteration Preparation Cycle

After the multiplicand is pre-normalized, the iteration preparation cycle that is identified by the iteration preparation trigger (Figure 5560) is set. During this cycle, the X12 multiple is generated, the exponents -64 are added together, the low-order multiplier group is decoded, the multiplier (J register) is gated to the register bus latch and back to the J register to accomplish a right four shift, the multiplier fraction (J register) is zero detected for a multiplier of zero, and the shift counter is set for use as an iteration counter.

The X12 multiple is generated by gating the contents of the K register (X16 multiple) to the normal input of the main adder and the contents of the M register (X4 multiple) right two to the complement input of the true/complement input of the main adder. The X4 multiple is subtracted from the X16 multiple, and the main adder output is gated to the L register; therefore, at the end of this cycle the L register contains the X12 multiple. The X6 multiple is obtained from the X12 multiple by gating the L register right one when needed.

The exponent sum -64 is obtained by transferring the exponent register to the true/complement input of the exponent adder and the multiplier exponent (J register 0-7 MER, MDR, and ME even address or J32-39 ME odd address) to the normal input of the exponent adder. Since the operand exponents are excess 64 numbers, the AEOB position 1 is complemented to produce the exponent sum less 64. The exponent sum is gated into the exponent register.

The low-order multiplier group is gated to the multiplier decoder (Figure 5048), which selects a multiple gating trigger. The multiplier group is located in J register bits 27-31 for the ME (even address) and MER instructions, and in bits 59-63 for the ME (odd address), MD, and MDR instructions. The low-order multiplier group is decoded as though the low-order bit of the group is a zero. If the low-order bit is a one, K register bits 0-59 are gated right four (X1 multiple) to the main adder during the next cycle with the decoded multiple.

To shift the multiplier right four with respect to the multiplier decoding circuits, the J register (Figure 5061) is gated left four to the register bus latch. The register bus latch is gated right eight to the J register to bring the next multiplier group into place. During this transfer to the register bus latch, the J register exponent is set to zero for proper multiplier decoding during the last iteration cycle.

The shift counter is set to a value of 6 if the instruction is a single precision or to 14 if the instruction is a double precision instruction. For each multiply iteration cycle, the shift counter is decremented by one.

During the first iteration preparation cycle, J register positions 8-31 [MER, ME (even address) instructions] or positions 8-63 [MD, MDR, or ME (odd address) instructions] are zero detected. If the multiplier is zero, a zero product results and the test cycle trigger is set. The K register and

the exponent registers are set to zero, the E last cycle trigger is turned on, and the zero result is gated into the first operand location. If the multiplier is not zero, the iteration preparation trigger is set again with the iteration trigger to identify the cycle in which the first multiply iteration occurs.

Iteration Cycle

The multiply iterations are identified by the iteration preparation trigger (Figure 5560) and the iteration trigger (Figure 5561) being on. The iteration trigger being on indicates a cycle in which a decoded multiple is added to or subtracted from the partial product. The iteration cycle is identical to the fixed-point multiply iteration; the decoded multiple gates either the M register or the L register to the true/complement input of the main adder and the K register bits 0-59 (partial product) are gated right four to the normal input of the main adder when the iteration trigger is on. Whether the input to the true/complement input of the main adder is complemented or not depends on the high-order bit of the multiplier group; if it is a zero, a true add cycle is taken or if the high-order bit is a one a complement add cycle is taken and the main adder hot 1 trigger is set.

At the beginning of the first iteration, the K register contains the X16 multiple, thus the K register (positions 0-63) is gated right four to the main adder during the first iteration cycle if the low-order bit of the group is decoded as a 1. The new partial product, located in the main adder out latches, is gated into the K register. If the multiple is a two cycle iteration, the iteration preparation trigger is turned off, the next multiple is not decoded until the X2 or X6 multiple is added or subtracted from the partial product, and the add trigger is turned on to identify the cycle in which the X8 multiple is added to or subtracted from the partial product.

Add Cycle

The second add cycle or any two-cycle iteration is identified by the add trigger (Figure 5566) being turned on. During the add cycle, the M register positions 0-63 are gated right one (X8 multiple) to the true/complement input of the main adder and the K register positions 0-63 are gated straight (partial product plus or minus the X2 or X6 multiple) to the other input of the main adder. The new partial product is gated from the main adder out latches to the K register. If the multiplicand is a complement number, 1 bits must be inserted at the high-order positions of the main adder true/complement input, which are vacated because of the multiple generation. If the multiplier is a complement number, the last multiplier group must be decoded as though the high-order bit of the group is a 1 bit.

During each iteration cycle, the shift counter is gated to the true/complement input of the exponent adder and one is subtracted from it. The result is returned to the shift counter. When the last iteration cycle is taken, the put away trigger is set to identify the put-away cycle.

PA Cycle

During an iteration preparation cycle in which the remaining multiplier digits are decoded as zeros, the iteration trigger and either the iteration preparation trigger or the add trigger is on. At the end of the cycle the partial product is gated to the K register and the M register. The product may have one leading zero digit because the multiplicand is digit normalized but not bit normalized.

The shift counter may not be equal to zero when the remaining multiplier groups are zero. If the shift counter is not zero, it contains the number of leading zero digits in the multiplier fraction because it was originally set to a value equal to the total number of multiplier digits.

The PA cycle is used to store the product and exponent in the specified floating-point register. This cycle is identified by the put away trigger (Figure 5554) being set. Bits 0-55 of the K register are transferred to bits 0-55 of the floating-point register specified by R1 or the instruction and the exponent register bits 0-7 are transferred to bits 56-63 of the floating-point register.

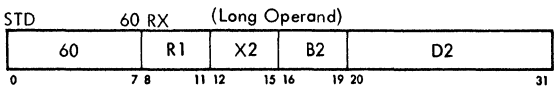
The product is not valid if the shift counter is not zero or if the product fraction is not normalized. If the fraction is not normalized, the M register (Figure 5067) is gated left four to the true/complement input of the main adder. The result is returned to the K register and the M register. Also, if the result is not normalized or the shift counter is not equal to zero, the shift counter is gated to the true/complement input of the exponent adder, the exponent register is gated to the normal input of the exponent adder, and the contents of the shift counter is subtracted from the contents of the exponent register. The AEOB position 1 is complemented because exponents and shift counter values are excess 64 numbers in floating-point multiply instructions. The result is placed in the exponent register. If the product is not normalized, the exponent adder hot 1 trigger is not set, thereby reducing the product exponent by the shift counter value plus one. At the next A pulse the shift counter is set to zero. The put away trigger is set along with the E last cycle trigger for the next cycle.

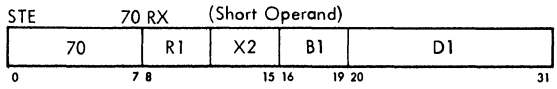
ELC Cycle

The E last cycle trigger (Figure 5555) is set if the result is not normalized or the shift counter is not equal to zero during the first attempt at gating the fraction and exponent to the floating-point register addressed by R1 of the instruction. The E last cycle trigger is not turned on with the put-away trigger if exponent underflow occurs and the underflow mask bit is a zero. The test cycle trigger is set to indicate that the K register exponent register is being set and the zero product is put away. If exponent underflow occurs and the underflow mask bit is a one, the test cycle trigger is not set. The correct sign and fraction are put away. The result characteristic is 128_{10} greater than the correct characteristic.

STORE

The store instructions are:





The store instructions are handled as follows:

1. The first operand is stored at the second operand location.
2. The first operand is not changed.
3. The STE instruction does not use bits 24-55 of the first operand.
4. The BCU is aware of the storage request by the beginning of E time.

Initial Operand Location

At the beginning of the execution cycle, the operand is located in the following registers by the methods described at the beginning of this chapter and by the gating shown on Figure 5079:

Instruction Format	R1 Operand
RX single (even address)	J0-31
RX single (odd address)	J0-31
RX double	J0-63

Instruction Sequencing

- Instruction sequencing is controlled by three triggers.
- The store preparation cycle is identified by the first FLP trigger.
- The store and wait cycle(s) are identified by the store trigger.
- The termination cycle is identified by the E last cycle trigger.

Figure 6414 is the data flow of the store instructions and Figure 6415 is the logic flow of the store instructions.

First Floating-Point Cycle

The store preparation cycle is identified by the first floating-point trigger (Figure 5552) being on. The operand (floating-point register addressed by R1 of the instruction) is located in the J register. During this cycle, the operand is gated from the J register by the logic on Figure 5061. If the store address is odd, the operand must be placed in the low-order half of the K register. In order to accomplish this, the output of the main adder is gated right 32 positions (Figure 5087) and set into the K register.

If the store address is even, the operand is placed in the high-order half (bits 0-31) of the K register. This is accomplished by gating the first operand from the J register to the main adder, and taking the output of the main adder and gating it into the K register.

Store Cycle

The instruction termination cycle is a variable-cycle operation identified by the store trigger (Figure 5567) being turned on. When the execution unit is started, a storage request cycle is also requested at the I to E transfer, and this variable cycle operation depends on the accept signal from the bus control unit. If, at the end of the store-preparation cycle, the accept signal is not received from the bus control unit, a wait cycle is taken. Wait cycles are taken until the accept signal is received from the bus control unit. During the wait cycle(s), gating is not performed and the operand is not modified.

ELC Cycle

The E last cycle trigger (Figure 5555) is turned on when the accept signal is received from the bus control unit. The accept signal gates the K register to the storage bus in, and the E last cycle trigger terminates the store instruction.

VARIABLE FIELD LENGTH

INTRODUCTION

CONCEPTS OF VFL

- VFL instructions process one data byte at a time.
- VFL operands may vary in length and need not conform to word boundaries.
- VFL operands and results are contained in storage.

The variable field length (VFL) feature of the System/360 Model 75 enables the execution of instructions in which the operands in storage may vary in length and be located in any byte addressable storage position. The operands need not conform to word boundaries; they may be contained within one or several storage words.

The E unit of the CPU contains circuits and units that provide byte gates, data paths, and controls that enable the execution of VFL instructions. Through the VFL circuits, the operands of each VFL instruction are processed one data byte at a time, serially.

VFL instructions are those that conform to the SS instruction format, and certain other fixed sequence VFL instructions in the RX and SI format.

SS instruction processing is storage to storage, with both operands and the result contained in storage. SS instructions are classified as either decimal or logical. Decimal instructions are those that perform decimal arithmetic, such as add, subtract, multiply and divide. SS logical instructions are those that perform logical functions with alphabetic or numeric data, such as edit, translate, and move. The move instruction, for example, moves an alphameric data field from one storage location to another without changing the data.

The fixed sequence VFL instructions are essentially FXP instructions that process one data byte through the VFL circuits.

Because the operands and result of SS instructions are contained in storage, the execution sequencing of SS instructions differs from those of all other instructions. The I unit does not prefetch operands for the SS instructions, nor does it continue instruction preparation during the SS execution. When an SS instruction is encountered in the instruction stream, the I unit prefetches all elements of the instruction words from storage then transfers execution functions to the E unit. Thereafter, the operands are fetched from storage to the K and L registers. Selected data bytes are gated, one at a time, from K and L registers to the VFL circuits (Figure 2040). The result bytes are returned to the K register. When all bytes of the operands are processed, the result in the K register is placed in storage and the instruction is terminated. The termination of the SS instruction releases the I unit to continue instruction preparation.

Instruction Format

- SS instructions occupy three halfwords in storage.
- Defines decimal or logical instruction.
- Defines length of both operands and their storage addresses.

The SS instruction format occupies three storage halfwords, 48 data-bits plus parity, and conforms to the format shown in Figure 24. Each SS instruction contains an operation code and sufficient data to define each operand in storage.

Operation Code

Bit positions 0-7 of the instruction contain the operation code in eight-bit binary form. The eight-bit operation code is commonly recorded as two hexadecimal digits. Figure 25 shows the operation code bit structure and hexadecimal representation for each of the SS instructions.

Storage Addressing

When an SS instruction is started and at various times during the execution, the storage location of each operand is referenced. The B1, D1, and L1 fields of the instruction define the storage location and length of operand 1. B2, D2, and L2, likewise, define operand 2.

The four-bit B1 or B2 instruction field specifies, in binary coding, one of 15 general registers in which the base address is contained.

The 12-bit D1 or D2 instruction field contains, in binary coding, the number of bytes the operand is displaced from the base address. When a storage reference to an operand is made, the storage address must first be computed. When a storage reference to operand 1 is made, the storage address is computed by adding the contents of the general register specified by the B1 instruction field to the displacement factor in the D1 field, B1 + D1. The storage address of operand 2 is similarly computed using the B2 and D2 instruction fields. If the B1 or B2

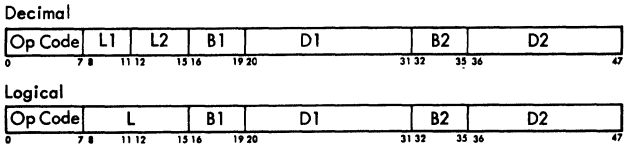


FIGURE 24. SS INSTRUCTION FORMAT

instruction field contains zero, then that address factor is considered to be zero and the D1 or D2 instruction field becomes the actual storage address; in this case, the contents of general register 0 is ignored.

The four-bit L1 or L2 operand length field defines, in binary digits, the number of data bytes operand 1 and operand 2 contains. For the majority of VFL decimal instructions, the maximum length of each operand is 16 bytes, or 32 decimal digits. However, for SS logical instructions, the L1 and L2 fields are combined as one to provide a maximum operand length of 256 bytes, or 32 storage words.

Either decimal operand may start at any byte location in storage, and extend into as many as three storage words. The sum of the base address and displacement factor, B1 + D1, or B2 + D2, defines the low-order storage address of the operand. When the length factor is added to the sum of the base address and displacement, B1 + D1 + L1 or B2 + D2 + L2, the high-order storage byte address of the operand is defined. Because each storage word contains 8 bytes, and each operand may be as many as 16 bytes long, the operand may be contained in one, two, or three storage words. The starting byte address, B1 + D1 or B2 + D2, and operand length, L1 or L2, determine the number of storage words that contain the operand. Figure 26 shows some of the starting byte to operand length relationships. An operand that is 8 bytes long can be contained in one storage word if the sum of B1 + D1 or B2 + D2 address byte zero of the storage word (Section A of Figure 26). An operand that is two bytes long can be contained in two storage words if the sum of B1 + D1 or B2 + D2 addresses byte 7 of the storage word (Section B of Figure 26). The majority of the decimal instructions limit the operands to a maximum length of 16 bytes, the equivalent of two storage words. However, an operand longer than 9 bytes can occupy three storage words, depending on the starting byte address of the operand (Section C, Figure 26).

VFL instructions are executed by stepping through the operand fields and processing one byte at a time. Decimal instructions, except divide, start execution with the storage byte that contains the low-order, least significant decimal digit (highest storage address), and steps toward the high-order end of each operand until all bytes are processed. Logical instructions start execution with the storage byte that contains the high-order, most significant, digit and steps toward the low-order end of the operands as each byte is processed.

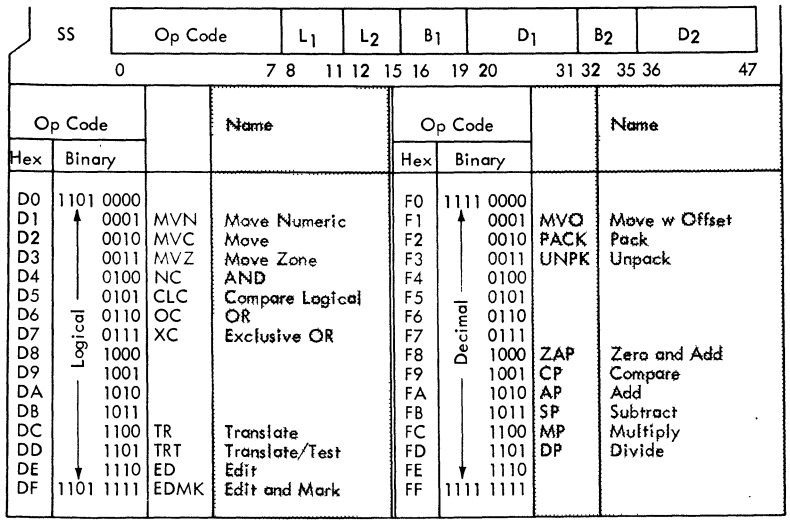
Because the decimal and logical SS instructions step through operands in opposite directions, a different sequence of reference to storage words is used. For example, if either operand of a decimal instruction extends across a storage word boundary, then the first storage word used is the one located at the higher storage address. The opposite is true for the SS logical instructions.

Data Format

Decimal data contained in storage is in the binary-coded decimal (BCD) format. A binary-coded decimal digit is represented by four storage data bits. Four data bits can represent any binary sum within the range of 0 to 15. A decimal digit can be any number within the range of 0-9. Therefore, the four-bit configuration that represents a binary sum within the range of 0-9 also represents decimal digits 0-9. The four-bit configurations with binary sums greater than 9 are used to represent the sign of a BCD digit, or the zone coding of alphabetic or special characters, Figure 27 and 28.

Decimal data is contained in storage in either of two formats, zoned data in the unpacked format, or decimal numbers in the packed format.

Unpacked Format: In the unpacked format, each decimal number is zoned. One zoned decimal digit occupies each byte of the decimal field in storage. In each byte, bits 0-3 contain the zone code and bits 4-7 contain the decimal number (Figure 29).



Op Code Hex	Op Code Binary		Name	Op Code Hex	Op Code Binary		Name
D0	1101 0000			F0	1111 0000		
D1	0001	MVN	Move Numeric	F1	0001	MVO	Move w Offset
D2	0010	MVC	Move	F2	0010	PACK	Pack
D3	0011	MVZ	Move Zone	F3	0011	UNPK	Unpack
D4	0100	NC	AND	F4	0100		
D5	0101	CLC	Compare Logical	F5	0101		
D6	0110	OC	OR	F6	0110		
D7	0111	XC	Exclusive OR	F7	0111		
D8	1000			F8	1000	ZAP	Zero and Add
D9	1001			F9	1001	CP	Compare
DA	1010			FA	1010	AP	Add
DB	1011			FB	1011	SP	Subtract
DC	1100	TR	Translate	FC	1100	MP	Multiply
DD	1101	TRT	Translate/Test	FD	1101	DP	Divide
DE	1110	ED	Edit	FE	1110		
DF	1101 1111	EDMK	Edit and Mark	FF	1111 1111		

Logical (D0–DF) / Decimal (F0–FF)

FIGURE 25. SS INSTRUCTIONS

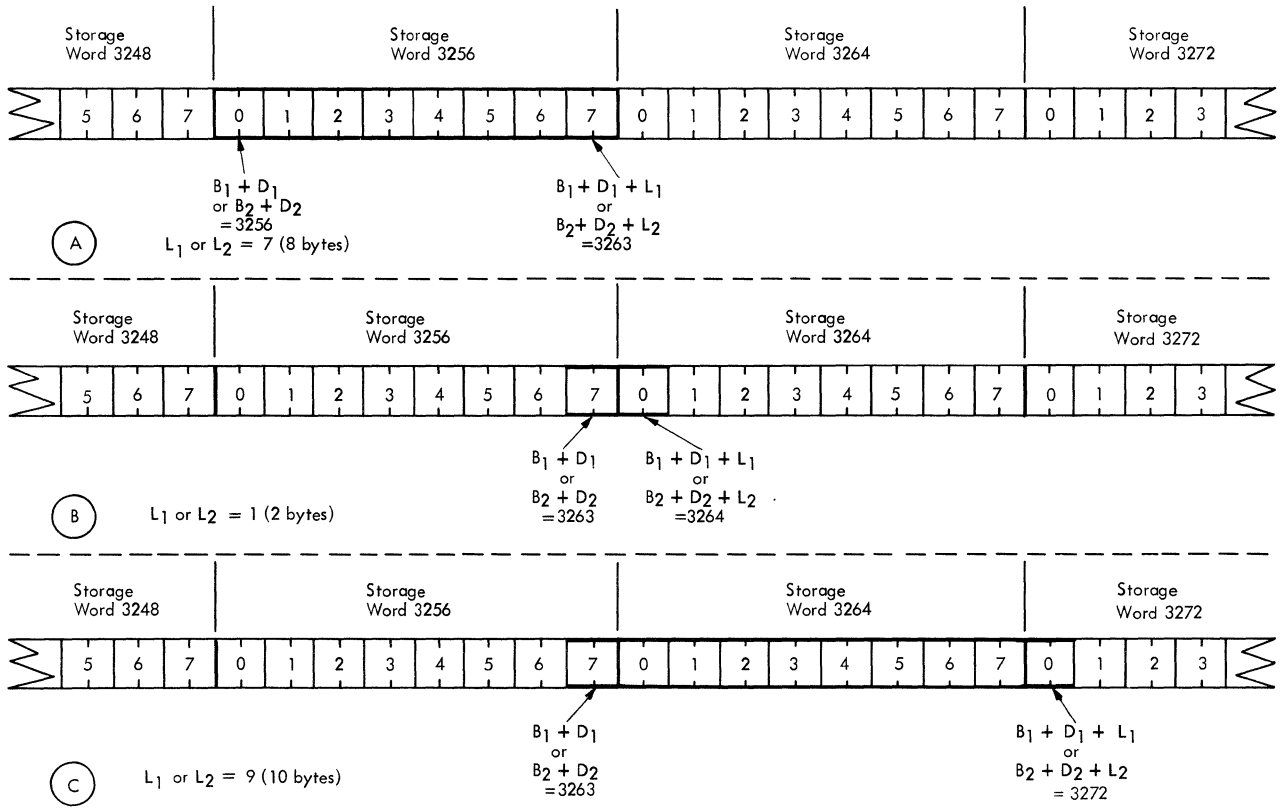


FIGURE 26. OPERAND LENGTH -- WORD BOUNDARY RELATIONSHIP

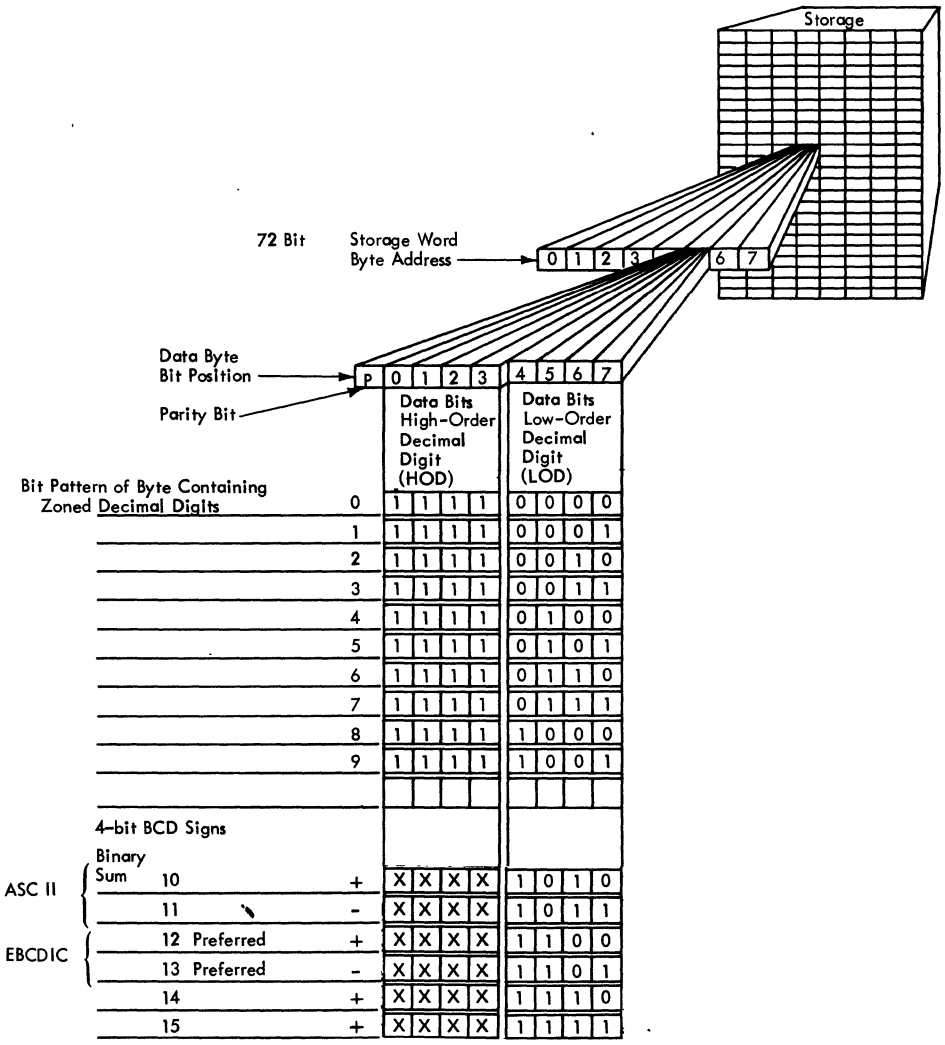


FIGURE 27. DECIMAL BYTE

Packed Format: In the packed format, each storage byte of the decimal field can contain two decimal digits.

All instructions that perform decimal arithmetic require that the data be in the packed format and the sign of the operand in the byte that contains the low-order decimal digit (the byte at the high-order storage address). See Figure 29.

Data may be changed from one format to another by use of the pack or unpack instruction.

VFL Instructions

- SS instructions process multiple bytes
- Fixed sequence instructions process single data bytes.

VFL instructions processed on System/360 Model 75 consist of all SS format instructions (Figure 25) and certain fixed sequence instructions that process a single data byte through VFL data paths.

Figures 9450 through 9457 are reference charts that show rules, conditions and examples of all SS instructions except the translate and edit instructions.

The fixed sequence VFL instructions are:

RX Format		
	STC	Store character
	IC	Insert character
SI Format		
	WRD	Write direct
	RDD	Read direct
	TM	Test under mask
	MVI	Move
	TS	Test and set
	NI	AND
	CLI	Compare logical
	OI	OR
	XI	Exclusive OR

The programming rules and objectives of the fixed sequence VFL instructions and the translate and edit instructions are contained in the publication IBM System/360 Principles of Operation, Form A22-6810 or see the Theory of Operation section of this manual.

VFL Data Flow

- Operands are fetched from storage to the K and L registers.
- Bytes from K and L are gated through VFL circuits to the K register.
- Contents of the K register are returned to storage.

Variable field length (VFL) data handling is designed as a subsystem within the main execution unit. The K and L registers are used as temporary storage for 64-bit words, plus eight parity bits. Operands are brought from main storage through the J register to the K or L registers, Figure 30. The K register is used for operand 1 (Op 1) and the L register for operand 2 (Op 2). The result of an operation is placed in the K register and at the proper times, the bytes of the K register that were changed are put back in the main storage.

The following text briefly describes the functions and controls of the various VFL functional units shown in Figure 2040. A complete description of the VFL functional units is contained in 2075 Processing Unit, Vol. 1 FEMI, Form 223-2872.

Right Byte Gate (RBG)

All 72 bits from the K and L registers are brought into the right byte gate (RBG). Two gating triggers, gate K with S or gate L with S, determine whether the K or L register is gated by the S pointer. The value in the S pointer determines which byte, 0-7, of the K or L register is gated through the RBG. The forcing of parity to the right side is OR'ed at the output of the RBG.

Right Digit Gate: The digit gate is between the RBG and true/complement/plus six (T/C +6) gate. This gate determines whether the two four-bit groups, 0-3 and 4-7, are straight gated or cross gated to the T/C +6 gate. The gate digit straight line is the inversion of the gate digits across line and, therefore, the output of the RBG is passed through the digit gate at all times. The gate digit across is used for pack, unpack, move offset, edit, edit and mark, and convert binary. The machine preferred zone and plus sign can be forced at the digit gate.

Left Byte Gate (LBG)

The 72 bits from the K register and 8 bits plus parity from DB/DC are brought into the left byte gate (LBG). Two gating triggers determine whether the K register is gated with the T pointer or DB/DC is gated through the LBG. The value in the T pointer determines which byte of the K register is gated through the LBG.

Decimal Adder

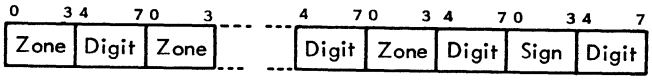
The VFL decimal adder is an eight-bit binary adder with modifications to the parity predict and output sums for decimal operations. Because VFL additions

Extended Binary-Coded-Decimal Interchange Code (EBCDIC)

BYTE Bit Positions 4567	00 00	00 01	00 10	00 11	01 00	01 01	01 10	01 11	10 00	10 01	10 10	10 11	11 00	11 01	11 10	11 11
0000	NULL				blank	&	-						>	<	‡	0
0001							/		a	j			A	J		1
0010									b	k	s		B	K	S	2
0011									c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	EOB	UC					f	o	w		F	O	W	6
0111	DEL	IDL	PRE	EOT					g	p	x		G	P	X	7
1000									h	q	y		H	Q	Y	8
1001					.		,	"	i	r	z		I	R	Z	9
1010					?	!		:								
1011					.	$	,	#								
1100					←	*	%	@								
1101					(	)	⌒	'								
1110					+	;	_	=								
1111					‡	¢	±	✓								

FIGURE 28. BCD CODING

Zoned Decimal Number



Packed Decimal Number

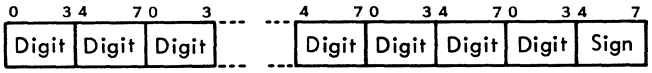


FIGURE 29. DATA FORMAT - UNPACKED--PACKED

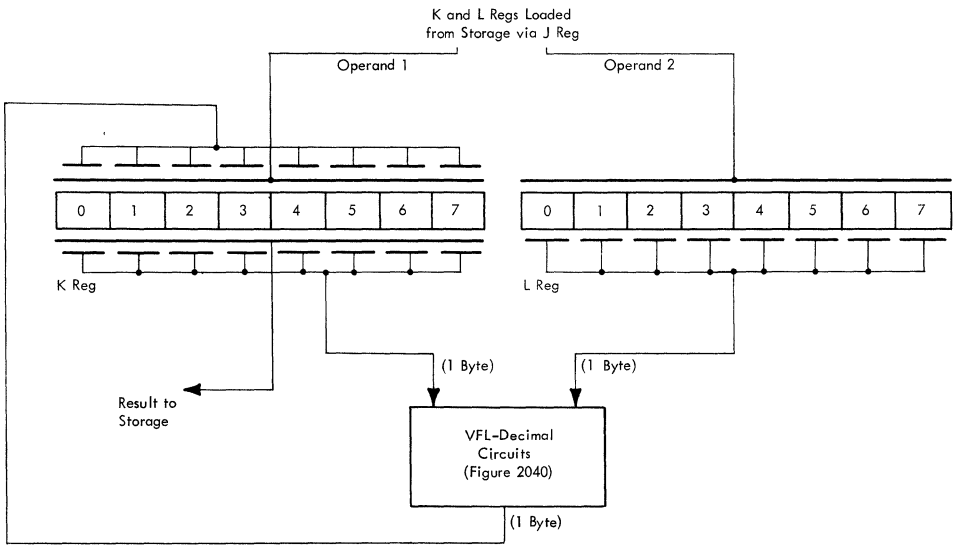


FIGURE 30. GENERAL DATA FLOW--MODEL 75 VFL

move through a field serially, adding 8 bits at a time, a carry out of the VFL adder is set into the carry trigger, which is gated back as a carry-in on successive byte additions. The VFL adder has full carry look-ahead on the digit level. The binary sums from the adder go into the decimal correction logic. This correction logic is latched and can be gated decimal and binary. Using excess-6 addition, the binary sum of two digits is the correct decimal sum, if there is a carry out of that digit position. If there is no digit carry, the binary sum is six higher than the correct decimal sum. The decimal correction latches are gated as:

1. Gate binary sum if not decimal add or digit carry.
2. Gate binary sum minus 6 if decimal add and not digit carry.

Note that the decimal correction is gated for both decimal add and subtract but the gate decimal true at the input to the adder is activated only for true decimal add.

TC + 6 Gate: The excess-6 method is used for decimal addition. This allows the VFL decimal adder to be a binary adder with adjustment on the input and output for decimal additions. Data are gated into the right side of the decimal adder in one of three ways:

1.	Binary True	BT
2.	Decimal True (TC + Six)	DT
3.	Complement	Compl

Only one complement gate is required since the 9's complement of a decimal number, plus 6, equals the 1's complement of the binary number. Therefore, the complement gate (1's complement) is good for both binary and decimal additions.

Decimal Digit	9's Compl	+6
0000	1001	1111
0001	1000	1110
0010	0111	1101
0011	0110	1100
ETC		

The input gates to the decimal adder are split for the high-order digit (bits 0-3, HOD) and the low-order digit (bits 4-7, LOD). Bit 7 has a separate complement control so that the machine preferred plus sign can be changed to the machine preferred minus sign.

Right Side Parity Adjust: The byte parity must be adjusted whenever a partial byte is gated through the adder or bits are altered as they are gated to the adder. When gating digits decimal true, decimal digits 4 and 5 are the only ones that change parity.

Two gating combinations of decimal true that require parity adjustment are:

1. (HOD DT). (LOD BT)
2. (HOD DT). (LOD DT)

All other parity adjustments are made because either HOD or LOD is not gated to the adder. Figure 31 shows the possible gate combinations on the adjusted parity.

Left Side Adder Input: The LBG is connected straight to the left side input of the decimal adder. This gate is split between bits 3 and 4 for high- and low-order digits. The parity adjust gate has the following combinations:

1. P straight
2. PH -- adjusted for HOD removed
3. PL -- adjusted for LOD removed

Forced Bits	Digit Gates	True-Complement-Plus Six	Adjusted Parity
Sign	ST	(HOD DT) · (LOD BT) · (Invert Sign)	(HOD=4/5) ⊻ $\overline{PL}$
Sign	ST	(HOD DT) · (LOD BT) · $\overline{\text{(Invert Sign)}}$	(HOD=4/5) ⊻ PL
	ST	(HOD DT) · (LOD DT)	(HOD=4/5) ⊻ (LOD=4/5) ⊻ Pin
Sign	ST	(HOD Compl) · (LOD BT) · (Invert Sign)	$\overline{PL}$
Sign	ST	(HOD Compl) · (LOD BT) · $\overline{\text{(Invert Sign)}}$	PL
	ST	(HOD Compl) · (LOD Compl)	Pin
	ST + CR	(HOD BT) · (LOD BT)	Pin
Zone	ST	(HOD BT) · (LOD BT)	PH
Zone	ST	(HOD BT)	Pin (Forced Parity)
	ST	(HOD BT)	PL
	ST	(LOD BT)	PH
	ST	Input on left side, no input on right side	Pin (Forced Parity)
	CR	(HOD BT)	PH

ST = Straight
CR = Cross
PH = Parity adjusted for removal of HOD
PL = Parity adjusted for removal of LOD

FIGURE 31. T/C + 6 GATE COMBINATIONS

Parity is forced to the left side of the adder to the parity adjust gate.

AND-OR-Exclusive OR-Mask

The AOE is used for logical connectives and the latch data path for storage protect key and direct data input sense lines. With one exception, the parity gated out with the AOE result byte comes from the parity generator on the output of AOE. The one exception is storage protect keys parity, which is gated back to the K register with the output of the AOE when executing an insert storage key instruction. Similar to an adder, the AOE has two inputs. One side has inputs from the LBG and Y-Z counters. The other side has inputs from RBG and direct data input sense lines. (The storage protect keys are combined with the direct data lines prior to their entry to AOE.) The output of AOE goes to the K register and to DB/DC. This connection to DB/DC is twisted so that high- and low-order digits are interchanged.

Digit Buffer (DB)

The digit buffer is a four-position register with positions numbered 0-3. Position 0 is the high-order position. The digit buffer holds the zone portion of the fill character for the edits instruction and high-order digit of quotient bytes for decimal divide. The digit buffer is used in conjunction with the digit counter to hold, for comparison to the dividend, the high-order 8-bits of the normalized divisor when executing fixed- and floating-point divides.

Input to the digit buffer are lined as follows:

Digit Buffer	0	1	2	3
LBG	0	1	2	3
AOE	4	5	6	7
Digit Counter	8	4	2	1

Output of the digit buffer are considered along with the digit counter described in the following text.

Digit Counter (DC)

The digit counter is a four-position counter used as a counter in decimal multiply and divide, and as a temporary storage register in edit, edit and mark, pack, unpack and move with offset instructions. Inputs to the digit counter are:

Digit Counter	8	4	2	1
VFL	4	5	6	7
LBG	4	5	6	7
AOE	0	1	2	3
Multiplier Bus	0	1	2	3
Force 9 to DC	1	X	X	1

The DC is a trigger register connected to a latched incrementer that feeds back into the register. The incrementer can be either increased by one or decreased by one.

The parity bit associated DB/DC is set from input parity when one is available. (Multiplier bus does not have a parity bit.) The DB/DC parity bit can also be sent from a parity adjuster when counting DC up or down. The input is used only with decimal divide when the correct parity is available at the beginning of a count.

The outputs of DB and DC are taken together as a byte to either LBG or the byte distribution (TD in) to the K register.

S and T Pointers

The S and T Pointers are three-position counters that are used as byte address pointers in the SS and SI format instructions. The one input to each pointer is the H register bits 21-23. For SS instructions,

the starting byte address for operand 1 (Op 1) and operand two (Op 2) is placed in T and S respectively. Each pointer is stepped up or down one as each byte is processed depending on the direction of movement through the operand field.

Each pointer has a three-position trigger register with a latch output, implemented with an incrementer-decrementer. The incrementer-decrementer modifies the output of the latch to give the next higher or lower register value, depending on whether the pointers are being stepped up or down.

Three decoders decode the output of the pointers and provide eight address lines to control the byte gating. These decoders are:

1. S out decoder -- connects to the S register and points to a byte of the L or K register for the RBG.
2. T out decode -- connects to the T register and points to a byte of the K register for the LBG.
3. T in decode -- connects to the T latch and controls the release of the K register byte when gating the result byte to the K register.

The pointers are also decoded for 0 or 7 to determine when main storage word boundary is reached.

Y and Z Counters

The Y and Z counters are each four bits in length. They are used as two four-bit counters for SS decimal instructions, and as one eight-bit counter for SS logical instructions. In SI and direct data instructions, Y and Z are used as an eight-bit register to hold operation code bits 8-15. Operated individually, Y or Z can be counted up or down by ones. Operated as an eight-bit counter, they can be counted up or down by one or by eight. Counting by eight is used for the move instruction in transmit mode.

Each counter is a four-bit register feeding through an incrementer-decrementer to a four-bit latch. The latch is connected back to the register input.

When used as counters, in the execution SS instructions, Y and Z determine when the operation is complete. Figure 32 shows the starting and ending conditions for all SS instructions.

The Y and Z counters have outputs to control decoders, AOE and direct data gate. The connection to AOE is for the immediate instructions (SI).

The direct data gate places the contents of Y and Z (operation code bits 8-15) on eight lines for three machine cycles in executing a read direct or write direct instruction.

The Y and Z counters are set from IOP (8-15) at B time of the last cycle of every instruction. This allows the I unit to overlap T1 and T2 with the SI and byte type RS and RX instructions. This set of Y and Z is similar to the set of EOP.

Instructions	Start Y	Start Z	Count Y	Count Z	Count Y and Z	End Op
AP, CP, SP, ZAP	L1	L2	Dn	Dn		Y and Z = 0
MP	L1	L2	Dn	Dn		Y = 0
DP	L2	L2	Up	Dn		Y = L1
MVO, PK, UNPK	L1	L2	Dn	Dn		Y = 0
MVN, MVC, MVZ, CLC, NC, OC, XC	L				Dn	YZ = 0
TR, ED, EDMK	0				Up	
TRT	0				Up	YZ = L or Nonzero Character

FIGURE 32. END OPERATION CONDITIONS--VFL

Direct Data Register

The direct data register is an eight-bit register (no parity) that is set with a byte from main storage on a write direct instruction. The contents of the DD register remain fixed until another write direct is executed.

Multiplier Bus

The multiplier bus is a four-position gate used to transfer the low-order 4-bits (J60-63), without parity, from the J register into the DC during the execution of the decimal multiply instruction.

VFL EXECUTION AND CONTROL

- Execution of SS instructions is divided into five sequences:
  1. Set-up sequence.
  2. Iteration sequence.
  3. Store-fetch sequence.
  4. Prefetch sequence.
  5. Address put-away (TRT and EDMK only).

- Set-up sequence fetches first word of both operands from storage and sets VFL control.

- Prefetch sequence fetches next operand 2 word if needed.

- Iteration sequence gates data through VFL units one byte at the time.

- Store fetch sequence. Stores completed results and fetches next operand 1 word.

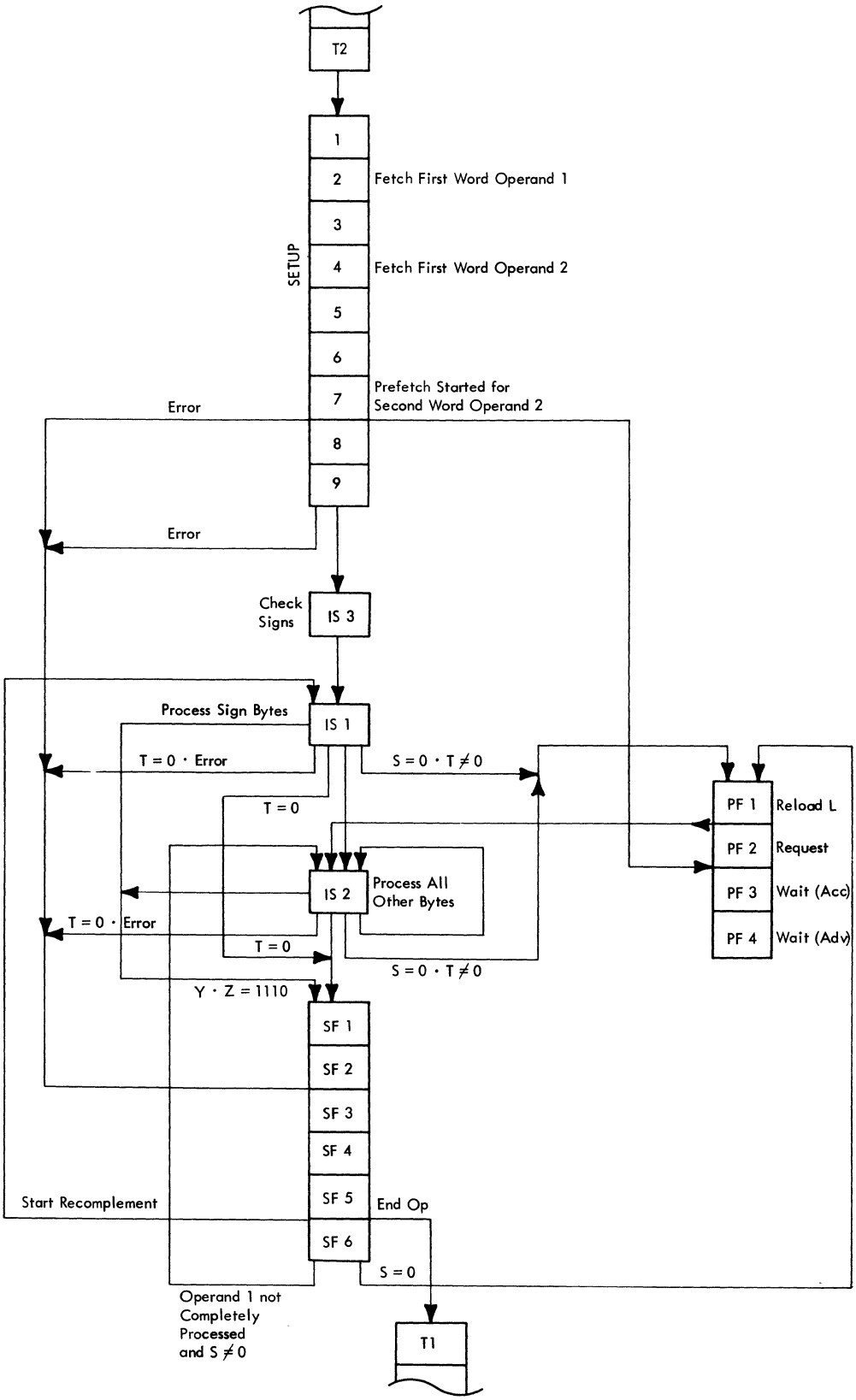


FIGURE 33. SS EXECUTION SEQUENCE — AP/SP

VFL Execution

The execution of all SS instructions starts with a set-up sequence. For repetitive byte operations, iteration sequences are used. Operand 1 fetching and storing is done by store-fetch sequences. Operand 2 fetching is done by prefetch sequences. For those instructions in which the result or address is put-away in a GR, the put-away is done by sequencer A, B, C, and D.

Figures 33 and 34 show an example of a decimal add sequence and how the sequences described in this section are used.

Set-Up Sequence

The set-up sequence prepares counters and registers to start the iteration sequences. This consists of:

1. Address calculation (low-order of 3 bits set into T and S) and set initial fetch request, for each operand.
2. Address comparison for overlapping fields.
3. Initial setting of Y and Z operand length counters.
4. Transfer operand words when they arrive from main storage, from J register to K and L registers.
5. Set initial values into ER and SC for logical instructions.
6. Set VFL gating triggers.
7. Start iteration sequence.

The address comparison for overlapping fields is made because byte operations must be executed in such a way that results appear to have been generated by operating one byte at a time from main storage. When the operands do not overlap into the same storage word, there is no difference between operating a byte at a time or eight bytes at a time from main storage.

If the difference between the starting address of the two operands is 0-7, there is a possibility that bytes of both operands will be taken from the same storage word during execution. When operand 1 and operand 2 are in the same storage word, the RBG is switched to gate from the K register instead of the L register, and thus, both the LBG and RBG take bytes from the K register (see "Overlap Control").

If the difference between the starting address of the two operands is 8-15, then the operand 2 word which would be fetched by a prefetch is presently in the K register. Therefore, the contents of the K register are transferred to M register during each store-fetch sequence, then M is transferred to L in the next prefetch.

In decimal operations, the comparison is made to determine if operand 1 resides in lower order storage than operand 2. In logical operations, the comparison is made to determine if operand 1 resides in a higher order storage location than operand 2.

Iteration Sequence

The VFL iteration sequence executes the arithmetic or logical functions specified by the instruction. In general, one data byte of each operand is processed each iteration cycle, controlled by one of three

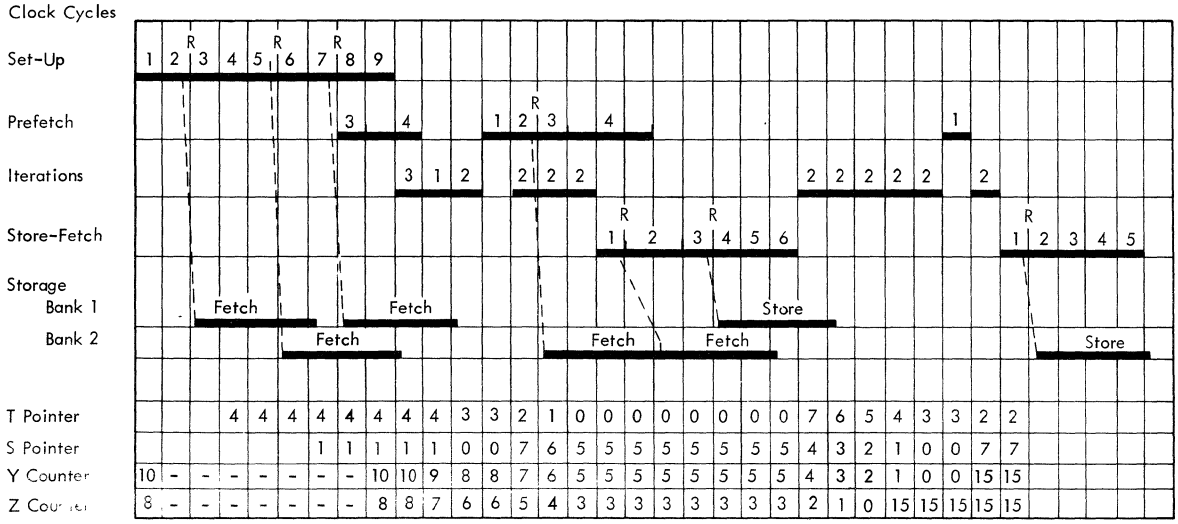


FIGURE 34. SS INSTRUCTION EXECUTION EXAMPLE--DECIMAL ADD

iteration sequencers, IS 1, IS 2, or IS 3. The precise function of each sequencer depends on the instruction in process. For example, during the execution of the decimal add (AP) instruction (Figure 34), IS 3 cycle checks the sign digit of the two operands and sets controls for the correct algebraic addition; IS 1 provides the proper gates to the decimal adder to add the low-order decimal digit (HOD of the byte) of each operand; thereafter, IS 2 cycles repeat to add one byte of each operand each cycle until all bytes are added.

If a word boundary of either operand is encountered during iteration cycles, the iteration sequence is suspended until a prefetch sequence or store-fetch sequence provides the next word; iterations then resume.

Each cycle that an Op 2 byte is processed, the Z counter and S pointer are stepped. When S steps down to zero for a decimal instruction or up to seven for logical instructions, an Op 2 word boundary is encountered and a prefetch sequence started to fetch the next operand 2 word. When Z steps down beyond zero to 15 (1111) all Op 2 bytes are processed and byte gating for Op 2 is terminated.

Each iteration cycle that an Op 1 byte is processed Y counter and T pointer are stepped. When T steps down to zero for decimal instructions or up to seven for logical instructions, an Op 1 word boundary is present, iteration cycles are suspended and a store-fetch sequence occurs. The store-fetch sequence stores the completed result word and fetches the next Op 1 word to be processed. Iteration cycles resume at the conclusion of the store-fetch.

Prefetch Sequence

When Op 2 is in more than one storage word a prefetch sequence is used to fetch the second and subsequent Op 2 words before they are actually needed. The fetch for the first word of Op 2 is made during the early cycles of the set-up sequence. If Op 2 is in more than one storage word, the first prefetch is started during set-up (SU 7 cycle in Figures 33 and 34). Thereafter, a prefetch sequence is initiated each time an Op 2 word boundary is encountered during iteration cycles.

Four sequencers are used to execute a prefetch, PF 1 through PF 4. The prefetch cycles occur in sequence and, except PF 1 cycles, overlap other execution cycles. PF 2 is the cycle in which the fetch request for the next Op 2 word is initiated; PF 3 is the wait cycle for the accept from BCU. It may span one or several CPU cycles depending on storage priorities. The prefetch sequence then waits in PF 4 cycles until the Op 2 word arrives from storage into the J register; the Op 2 word is then transferred from the J to the M register and the prefetch sequence terminates.

When an Op 2 word boundary is encountered during iteration cycles, the prefetch sequence starts with PF 1 cycle. PF 1 cycle transfers the previously prefetched Op 2 word from the M register to the L register. PF 2 cycles follow PF 1 if another Op 2 word must be fetched from storage; otherwise, PF 1 cycle transfers the last Op 2 word from the M register to the L register and terminates the prefetch sequence. Iteration cycles are suspended during PF 1 cycles; iterations are resumed after PF 1 and are concurrent with PF 2 through PF 4.

Store-Fetch Sequence

The store-fetch sequence is used to fetch the next Op 1 word to be processed from storage and to store the completed result from the K register. The store-fetch sequence is also used as the terminating sequence for all SS instructions except the TRT instruction.

The store-fetch sequence consists of six sequence cycles, SF 1 through SF 6 (Figures 33 and 34). In general, SF 1 and SF 2 control the fetch for the next Op 1 word and SF 3 through SF 5 control the request to store the result word contained in the K register. SF 6 is the cycle that waits for the new Op 1 word to arrive from storage if a fetch is started in SF 1.

Each time an Op 1 word boundary is encountered during execution a store-fetch sequence is initiated. The T pointer is used to control the gating of Op 1 data bytes; it is stepped up or down one as each data byte is processed. The crossing of an Op 1 word boundary is, therefore, indicated by the T pointer. When execution moves right to left through an operand field, the T pointer equals 0 at a word boundary. When execution moves left to right through an operand field, the T pointer equals 7 at a word boundary. When the execution moves to an Op 1 word boundary, iteration cycles are suspended; a fetch is made to get the next Op 1 word from storage; the completed result word is stored, and iteration cycles are resumed when the new Op 1 word arrives from storage. In this case, the store-fetch sequence starts at SF 1 and sequences through SF 6.

When all data bytes of both operands are processed, or an interupt is signaled, a store-fetch sequence is initiated to terminate the instruction. The data bytes are counted as they are processed during iteration cycles. When the number of bytes specified by the length field of the instruction have been processed, the instruction is terminated. The Y and Z counters are used to count the data bytes and signal the end of the instruction. For some VFL instructions, the Y and Z counters are set to the operand length in the IOP register, then stepped down during iteration cycles until Y and Z equals zero. Other instructions reset Y and Z to zero to start, then step

them up during iteration cycles until Y and Z equals the length contained in the IOP register. Figure 32 shows how the Y and Z counters are used and the end operation conditions of each VFL instruction.

Address Put-Away

Two instructions, translate and test and edit and mark, put information in general registers as part of their results.

Translate and test inserts the argument address (Op 1 address) into the low-order 24 bits of GR 1 and the translated byte (nonzero byte from the translation table, Op 2) in the low-order eight bits of GR 2. These results are inserted in GR 1 and 2 only if a nonzero byte is found.

Edit and mark inserts the byte address of the first significant result digit in the low-order 24 bits of GR 1.

Sequencers A, B, C, D, IS 1 and IS 3 are used for the TRT address put-away and sequencers A, B, C, D, and IS 1 are used for the EDMK address put-away.

Interrupts

VFL operation can have the following interrupts:

1. Invalid address
2. Data
3. Specification
4. Decimal overflow
5. Decimal divide check

Invalid Address: The invalid address interrupt can occur on any fetch and all SS instructions have at least one fetch. The address invalid trigger is reset at the beginning of each SS execution and then, once set, remains on even though valid words may return to the J register after the trigger is set. For all SS instructions, except multiply and divide, the address invalid is sampled at SU 9, and SF 6. For SS multiply and divide, when the address invalid trigger is on, the sequence is switched to SF 3 and terminates the instruction. The E interrupt trigger blocks the set of VFL request triggers during the SF sequence and causes the VFL end sequence trigger to be set.

Data Interrupt: Data are checked on each iteration cycle and the interrupt triggers are set when a sign or digit is detected in the wrong place. During the next store-fetch sequence, if the E interrupt trigger is on, the VFL end sequence is set and the setting of both VFL request triggers is blocked.

Specification Interrupt: The specification interrupt can occur on decimal multiply or divide. L1 and L2 are checked during SU 2 cycle. If L2 is equal to or greater than L1, or L2 is greater than 7, the store-fetch is set and the ending sequence follows.

Decimal Overflow: Decimal overflow can occur on AP, SP and ZAP. The occurrence of the overflow interrupts does not alter the execution of the instructions.

Decimal Divide Check: Divide check is sampled during sequence A of divide test sequence. A divide check switches the sequence to SF 3, which starts the end sequence and terminates the instruction.

VFL Control

VFL T1-8 Trigger: VFL T1-8 are a group of multi-purpose control triggers. All of these triggers are set at A time and VFL T2, VFL T3, and VFL T5, have latched outputs. The function of each trigger is controlled by the instruction being executed. (See Figure 9466)

VFL Store and Fetch Request Triggers: These are two intermediate request triggers used for E unit storage request. They have two outputs: one to the BCU, and one to the I Unit. In the I Unit, the store request trigger initiates an address compare and the fetch request to return the word to the J register. The BCU request triggers are set at the beginning of the cycle following that in which the E unit request triggers are set. The VFL request triggers are set at LB time and are reset with the A time and accept. The set for these triggers is latched to generate the gating line gate AA to SAR and H.

Store-Fetch and VFL Sequence Triggers: VFL sequence triggers 1-12 are dual-function triggers, control led by the VFL SF trigger. When the set-up sequence of an SS instruction starts, the SF trigger is off, and the VFL sequence triggers (1-9 or 1-12) control and gate set-up sequence functions. When the set-up sequence terminates, the SF trigger is set and thereafter, the VFL sequence triggers (2-7) control and gate functions for the store-fetch sequence.

Y-Z Counters: The Y and Z counters are the operand length counters for the SS instructions. The length counters start with the specified operand lengths and counts down to zero for all instructions except DP, ED, EDMK, TR and TRT.

Counting the Y and Z counters down maintains the count of the remaining operand bytes to be processed. The value in the length counter can be used to determine if another operand word should be fetched once a prefetch has started. The first cycle of the prefetch sequence transfers the previously

fetched word from the M register to the L register. If another word is needed (length counter shows more than eight bytes remain) the prefetch sequence continues to get the next word from storage.

For ED, EDMK, TR, and TRT there is no actual prefetch. The prefetch sequence is used to fetch operand 2 words but does not overlap iterations. For EDMK and TRT, the address of a byte in operand 1 is put in GR 1. The most convenient method in generating this address is to start with Y-Z at zero and count them up as operand 1 bytes are processed, then add Y-Z to B1 + D1 when the byte address is required. The end of the operation is indicated by Y-Z equal IOP 8-15.

In decimal divide, the number of quotient bytes to be generated is L1-L2. Therefore, L2 is set into Y and counted up until Y equals IOP (8-11).

When counting down, the counters are stepped with the set conditions for the iteration sequencers. Because the specified operand lengths are the number of bytes minus one, the counter value of all ones indicates the end of operation instead of a zero value (Y or Z counter is stepped down beyond 0 to 15). Furthermore, the counter latch is decoded instead of the register because the decoder is used to set and to reset triggers at A time. This means that the counter value of 1110 for decimal or 1111-1110 for logical operations indicates all bytes have been processed.

End Sequence Trigger: The VFL end sequence trigger is set by all SS instructions. With one exception, the VFL end sequence trigger is set two cycles before the end of the operation (Figure 35). The one exception is the translate and test instruction, which ends in an address put-away sequence. The set of the VFL end sequence trigger is also the VFL through signal to the I unit. The ELC is set for the last cycle on every SS instruction. This is done to take advantage of the built-in end operation control functions of the ELC trigger. Figure 35 shows the two end operation sequences.

VFL Zero Detect: The VFL data flow has two zero detects, one on the output of the digit gates (RBG ZD) and the other on the result bus back to K register (result ZD).

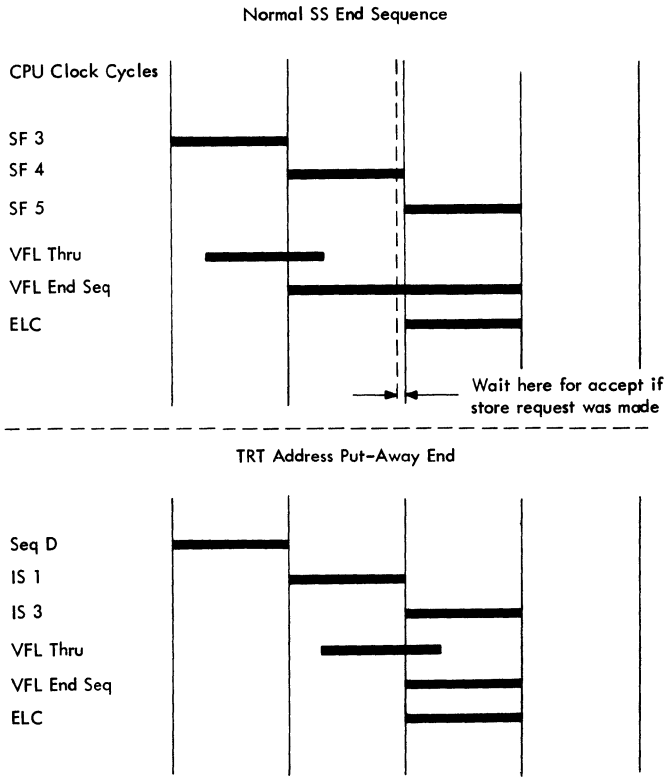


FIGURE 35. END OPERATION, VFL

The RBG ZD is connected to two latches; the low-order digit is latched for edit and both digits are ANDed and latched as a byte not zero latch. This is used in TRT and overflow detection.

The result ZD sets a trigger with a latch output. The zero detect logic has a control line to force zero in the low-order digit (sign position) for arithmetic operations. The trigger is actually set if a nonzero byte is detected and is in the off state at the end of any operation with a zero result. The result ZD is used by decimal arithmetic and logical compare operations.

Overlap Control

- Compare operand locations in storage.
- Set 0-7 or 8-15 overlap controls if operands overlap.
- Controls register to register data transfer during prefetch.
- Sets byte gate controls for iterations.

The two operands of an SS instruction may be located at any addressable storage location. Both may be contained within the same 72-bit storage word or separated by any number of storage addresses within the capacity of the system. The operands are permitted to overlap the same storage addresses unconditionally for some instructions and conditionally for others. For example, the move-with-offset instruction permits the operands to overlap in any manner, while those instructions that perform decimal arithmetic, such as add or subtract, permit the operands to overlap the same storage addresses only if the low-order bytes of each operand coincide.

When the starting bytes of the two operands are separated by a few byte addresses in storage, the gating of data bytes from each operand during iteration cycles may move into and out of common storage words as word boundaries are encountered. For example, consider a decimal instruction with Op 2 starting in an adjacent higher storage word than Op 1. During initial iteration cycles, Op 2 bytes are gated from the L register and operand 1 from the K register. If Op 2 crosses a word boundary before Op 1, both operands move into the same storage word in the K register; then both Op 1 and Op 2 bytes are gated from K register. Thereafter, if an Op 1 word boundary is encountered, iteration cycles are suspended while the data in the K register is transferred to the L register and the new Op 1 word is fetched from storage to the K register. When iteration cycles resume, Op 1 data bytes are gated from the K register while the remaining Op 2 bytes are gated from the L register.

Two overlap conditions are considered; if the starting bytes of both operands are separated by less than eight bytes in storage, 0-7 overlap exists, if the separation is more than 8 bytes but less than 16 bytes 8-15 overlap exists. A 0-7 overlap or 8-15 overlap control trigger is used for each of the overlap conditions.

One of the functions of the VFL set-up sequence is to determine the overlap status of the two operands and establish the most efficient byte gating control. During the early cycles of the set-up sequence, fetch requests are made to get the first word of each operand -- the storage word that contains the starting byte. The storage address of Op 1 is computed and the fetch made, then the storage address of Op 2 is computed and the fetch for the Op 2 word is made. Later, in the set-up sequence, the two storage addresses are compared to determine the number of bytes that separate them.

Through the main adder, the Op 1 storage address is subtracted from the Op 2 storage address and the result set into K register positions 0-31. K register positions 0-28 are zero detected and if equal to zero the starting bytes of the two operands are separated by less than eight bytes (one word) in storage. In this case, both operands may start in the same storage word or adjacent words. When K0-28 equals zero, the 0-7 overlap trigger is set; when K0-28 is not equal to zero and K0-27 equals zero, the 8-15 overlap trigger is set.

The K0-27 or K0-28 zero detection and the set of the appropriate overlap control trigger occurs during SU 8 cycle of the set-up sequence for all VFL instructions except for translates, edits, and multiply or divide. The set of the overlap control triggers are mutually exclusive; the set of 0-7 overlap resets 8-15 overlap and visa-versa.

Because the translate and edit instructions may not step through both operands serially or process data bytes of both operands at the same rate, the set and reset of the overlap control triggers occurs at various times throughout instruction execution. See instruction involved.

Overlap control is not used during the execution of decimal multiply or divide instructions. For the decimal divide instruction, all storage words of

both operands are fetched from storage and aligned in working registers before iterations start. Therefore, no prefetches occur during iterations and the store-fetch sequences only store. For decimal multiply, all storage words of Op 2 are fetched and aligned in the L register, and the first word of Op 1 is fetched and aligned in the J register during the set-up sequence. Therefore, no prefetches occur and the store-fetch sequence fetches the next multiplicand word and stores the completed product word.

0-7 Overlap: The set status of the 0-7 overlap trigger indicates that the starting addresses of the two operands are within eight bytes of each other. However, it does not indicate whether the operands start in the same or adjacent storage words. Earlier during the set-up sequence, the starting byte addresses of Op 1 and Op 2 are set into the T and S pointers respectively. T and S are compared to determine if the operands start in the same or adjacent storage words when a 0-7 overlap condition exists (Figure 36).

8-15 Overlap: The set status of the 8-15 overlap trigger signals that the two operands are separated by more than eight bytes but less than 15. Therefore, the two operands may start in adjacent storage words or be separated by one word as shown in Figure 37.

ER and SC Used as Word Counters

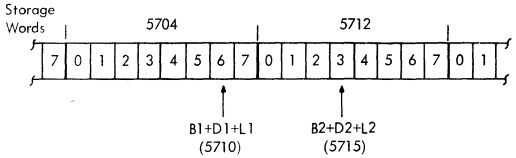
The exponent register (ER) and the shift count register (SC) are used to control storage addressing for those VFL logical instructions where the operand 1 and operand 2 data bytes are processed at the same rate. The prefetching of operand 2 data and the store-fetching of operand 1 data occur alternately. The ER maintains a word count of the result words processed; it is advanced one each time a result word is stored. The ER contains the correct increment to be added to the operand 1 starting address of B1 + D1 for the store address. The SC maintains the correct address increment to develop the next fetch address of either Op 1 or Op 2. The two tables of Figure 38 show the contents of the ER and SC for each prefetch and store-fetch operation.

In section A of Figure 38, an Op 2 word boundary is crossed before the first store-fetch of Op 1 is required. During the set-up sequence, prior to the execution of the instruction, VFL T5 control trigger is turned on if Op 2 will cross a word boundary before Op 1, in which case, the SC is set to two. In addition, during the set-up sequence, the first Op 2 word (B2 + D2 + 0) is placed in the L register and a fetch (first prefetch) is begun to place the second Op 2 word (B2 + D2 + 1) in the M register. The first prefetch that occurs after set-up addresses Op 2 data at B2 + D2 + 2. When a prefetch sequence starts, the SC is gated to the AA (line 2, Figure 39) to compute the fetch address, then the ER + 1 is placed in the SC (lines 11, 13, 14, and 15, Figure 39). The SC now contains the correct increment to fetch the next Op 1 word. At the beginning of the store-fetch sequence (Figure 40), the SC contains the correct increment to develop the fetch address and ER contains the store address increment.

After the fetch request has been initiated, the store address increment is transferred from ER to SC (lines 11, 13 and 15, SF 1 cycle in Figure 40). The store increment is used during SF 2 and SF 3 cycles (line 1 in Figure 40). During SF 3, ER is incremented by one to create the new store address increment. During SF 5 and SF 6, the SC is incremented two more than the ER and, therefore, contains the correct increment for the next fetch of Op 2 data.

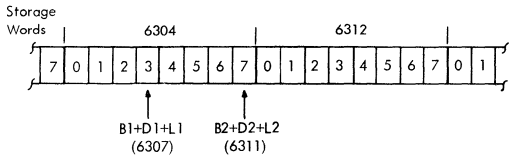
Section B of Figure 38 shows the Op 1 field reaching a word boundary before the Op 2 field. The SC is set to one during the set-up sequence (VFL T5 off), the correct increment for the first Op 1 fetch address. In SF 5 cycle of Figure 40, the SC is incremented to one more than the amount in the ER. This is the correct increment to address the third Op 2 word. During the next prefetch, Figure 39, ER plus one is placed in the SC during PF 2 to provide the address increment for the fetch of the next Op 1 word (used during the next SF). However, the increment amount in the SC remains the same as for the preceding fetch of Op 2.

Operand 1 and Operand 2 start in adjacent storage words



1. Initially, T > S and therefore, iterations start with "gate L with S" on.
2. When T = 3 and S = 0, a prefetch is initiated and "gate K with S" is set.*
3. When T = 0 and S = 5, a store-fetch is initiated, K is transferred to L and "gate L with S" is set.

Operand 1 and Operand 2 start in the same storage word

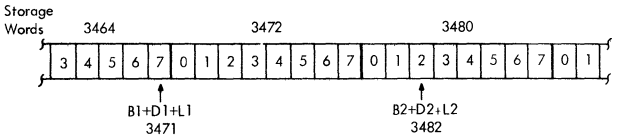


1. Initially, T ≤ S and therefore iterations start with "gate K with S" on.
2. When T = 0 and S = 4, a store-fetch is initiated, K is transferred to L and "gate L with S" is set.
3. When T = 4 and S = 0, a prefetch is initiated and "gate K with S" is set.

* "Gate L with S" and "gate K with S" are mutually exclusive. The set of one resets the other.

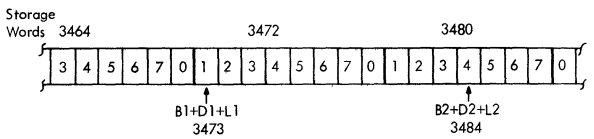
FIGURE 36. OVERLAP EXAMPLE - 0-7 OVERLAP

One word between starting points of Operand 1 and Operand 2



1. Initially, 3464 is in K, 3480 is in L, and 3472 is being prefetched to M. ("Gate L with S" is on and stays on).
2. When T = 5 and S = 0, a prefetch is initiated and M is transferred to L.*
3. When T = 0 and S = 3, a store-fetch is initiated and K is transferred to M.

Operand 1 and Operand 2 start in adjacent words

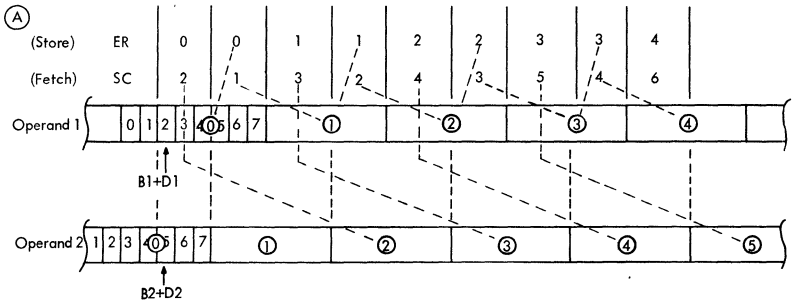


1. Initially, 3472 is in K, 3480 is in L, and 3472 is in L, and 3472 is being prefetched to M.
2. When T = 0 and S = 3, a store-fetch is initiated and K is transferred to M. The modified 3472. in K, goes into M on top of the prefetched 3472.
3. When T = 5 and S = 0, a prefetch is initiated and M is transferred to L.

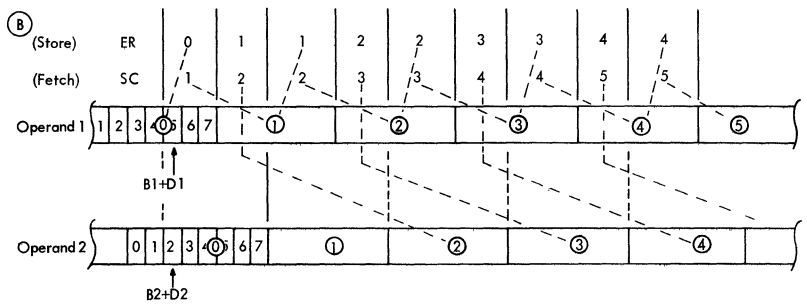
* With either of the overlap triggers on, prefetch is only one cycle, PF1.

FIGURE 37. OVERLAP EXAMPLES - 8-15 OVERLAP

Operend 2 ahead of Operand 1
Starting values: T=2, S = 5



Operand 1 ahead of Operand 2
Starting values: T=5, S=2



Strips labeled Operand 1 and Operand 2 represent groups of storage words in different storage locations.

FIGURE 38. ER AND SC WORD COUNTERS

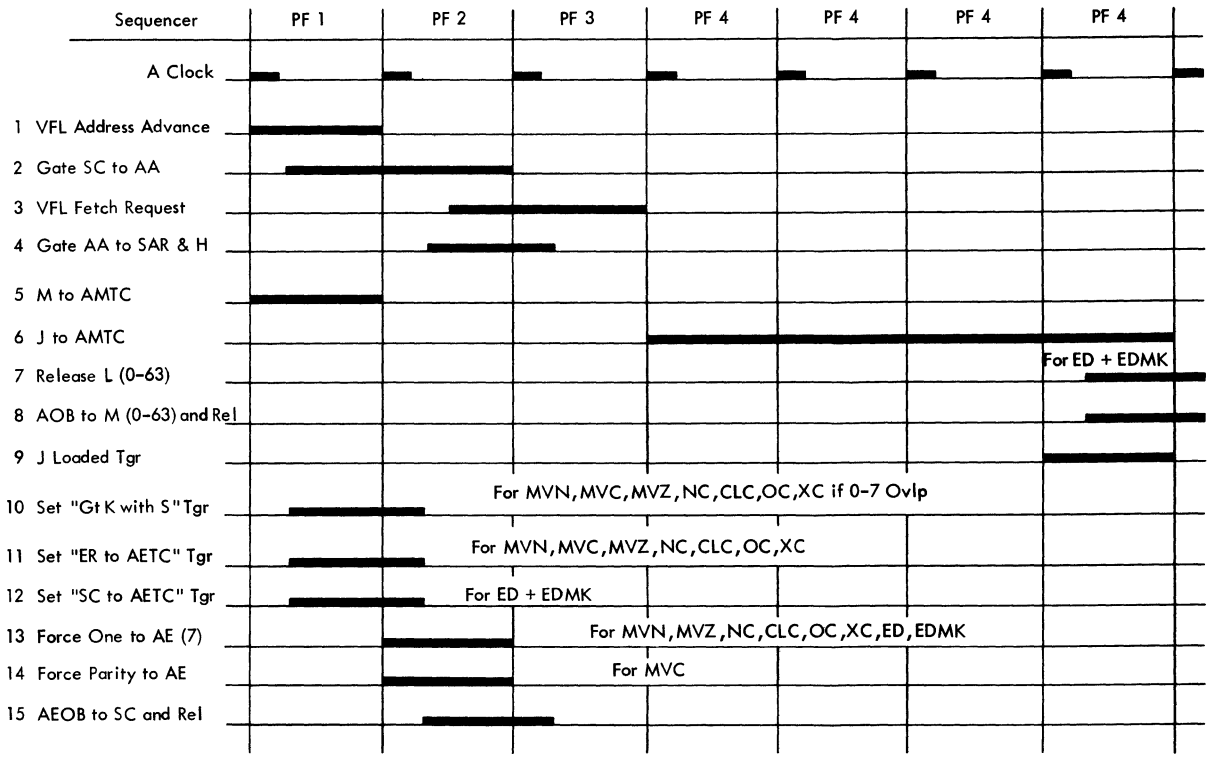
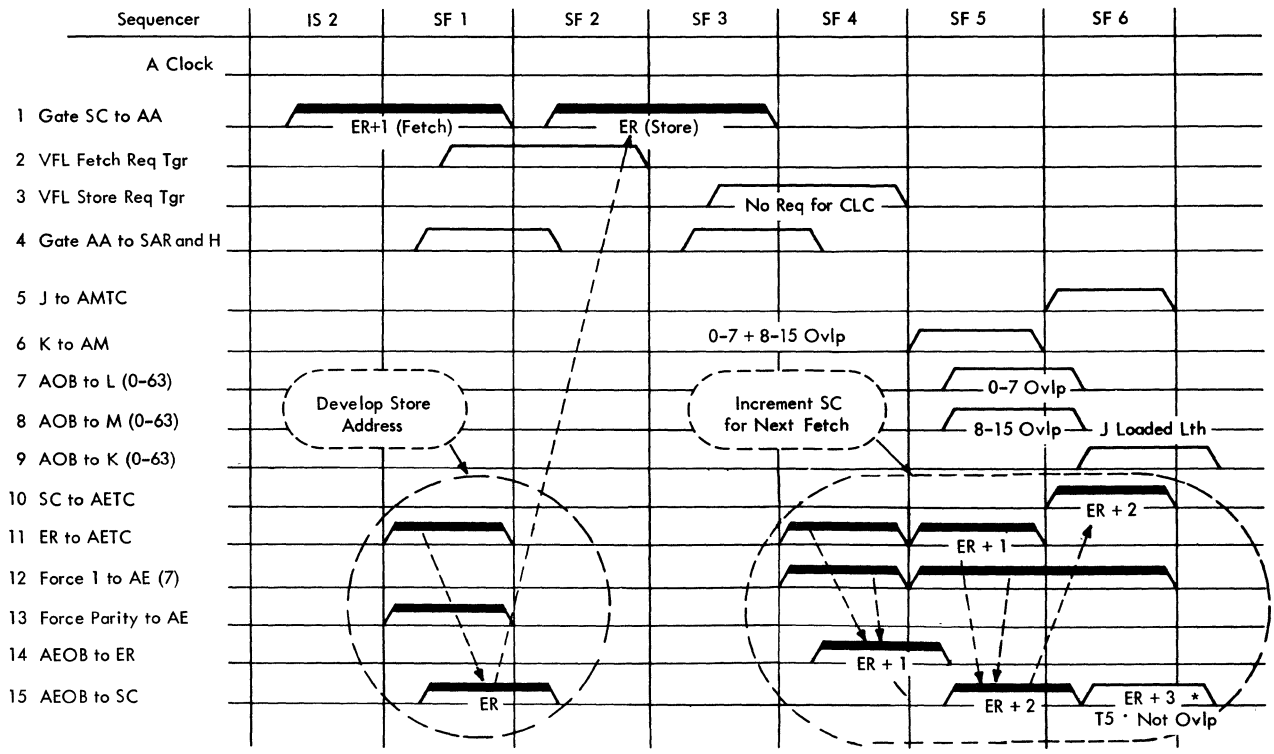


FIGURE 39. ER AND SC CONTROL DURING PREFETCH - LOGICAL INSTRUCTIONS (NOT TR OR TRT)



* T5 is set during set-up if S > T, which indicates operand 2 is crossing word boundaries ahead of operand 1.

FIGURE 40. ER AND SC CONTROL DURING STORE-FETCH - LOGICAL INSTRUCTIONS (NOT TR OR TRT)

VFL INSTRUCTION EXECUTION

- SS instructions start with set-up sequence -- fetch operands from storage.
- Iteration sequence follows set-up -- data bytes gated through VFL units.
- Prefetch sequence overlaps iterations -- fetches Op 2 storage words.
- Store-fetch sequence stores result word and terminates SS instructions.

This section provides a detailed description of the execution of VFL instructions; these include all instructions in the SS instruction format and certain other fixed sequence VFL instructions in the RX and SI format. In addition, convert and direct control instructions are included.

Where the functions of several instructions are similar, they are treated as a group. For example, decimal add, subtract, compare, and zero and add are similar in function and execution, and therefore, share common flow charts where applicable.

Because of the complexity of the decimal divide and decimal multiply instructions, they are explained separately.

Set-Up Sequence -- Decimal Instructions

- Fetch storage word that contains first byte to be processed for each operand.
- Compare storage address of each operand to determine if storage words overlap.
- Set starting byte address into S and T pointers.
- Start first prefetch if required.
- Set fetched words into K and L registers.
- Set VFL control and gating triggers.

The set-up sequence for decimal instructions pertains to the AP, SP, CP, ZAP, MVO, PK, and UNPK instructions. Prior to cycle by cycle functions, certain operand and storage address relationships are examined.

If Op 2 is contained in more than one storage word, a storage fetch request for the second word is made during set-up. This request is called the first prefetch. The length (L2) of the operand alone does not indicate how many storage words Op 2 may span. As an example, Op 2 could have only two bytes (L2=1) but start at byte address zero and be contained in two storage words. However, L2 could be seven with a starting byte address of seven and Op 2 contained in one storage word. Therefore, L2 is compared with S (starting byte address) to determine if the first prefetch should be initiated. If L2 is greater than S, Op 2 is in more than one storage word and the first prefetch is initiated during set-up. During iterations, a second prefetch is started when the first Op 2 word boundary is crossed (S=O). At this time, the length counter (Z) indicates the number of bytes that remain to be processed. If Z (L2) is greater than 7, the prefetch is continued, otherwise it is terminated after PF 1 cycle.

With the exception of pack and unpack, all decimal instructions that require detection or processing of overlapping fields move through both operands at the same rate. This means that the relative position of the two operands at the start of execution remains unchanged throughout the execution.

On unpack, the starting addresses are checked for an absolute difference of 0 to 7. If the difference is 0 to 7, the two low-order word address bits are updated in the exponent register and shift counter each time a word boundary is crossed. When these two partial addresses become equal, the crossing of the word boundary moves both operands into the same storage word and one register (K) is used for both operands.

When the difference of the starting addresses is 0-7, that is, when B2+D2+L2 minus B1+D1+L1 is less than 8, a comparison of the byte address indicates whether the two operands start in the same storage word. The starting byte addresses are in the S and T pointers, Op 2 in S and Op 1 in T. If S is less than T, the two operands start in adjacent storage words, the first Op 1 word is the second Op 2 word required. See Figure 41.

When operating in single cycle mode, the first set-up fetch request is made during SU 1 cycle (see Figure 6450). This allows the word that was returned to the J register to be transferred to the M register during SU 2 cycle. The rest of the set-up sequence is unchanged with exception of the start of prefetch. The start of prefetch is delayed to SU 9 cycle so that the word fetched does not return to the J register and destroy the first word of OP 2 which returned to the J register during SU 5 cycle.

Set-Up Functions

The following text describes the functions of SU cycles for decimal instructions, except for decimal multiply and decimal divide. The set-up sequence for multiply and divide are presented separately.

SU 1: Y is gated to AA with (SU L1 or SU T2) to calculate B1+D1+L1. The extended gate is due to the path length from VFL controls to the AA.

SU 2: The VFL fetch request trigger is set with the B clock. The output of this trigger goes to the I unit to indicate J as the return address and to the BCU to set their fetch request. The set of both VFL request triggers (fetch request and store request) is latched to generate the gate of the AA to SAR and H.

SU 3: Z is gated to the AA with (SU L3 or SU T4) to calculate B2+D2+L2. The VFL address advance line is up during cycle three so that B2 and D2 will be in IOP and gated to the AA during cycle four. The low-order three bits of H are gated to the T latch and T is released with SU L3. This puts the starting byte address of operand 1 in the T pointer. H(0-23) is gated to the incrementer. The latched output of the incrementer and incrementer extension is gated to K (0-31). Because nothing is gated into the incrementer extension, its output is zero with correct parity. The AOB (32-63) is gated to K (32-63) at the same time to put zeros with correct parity in the low-order half of K.

The sequence is held up here until an accept is received from the BCU. If an immediate accept is received, SU 3 takes only one cycle. This prevents a second request being made in SU 5 without an accept from the first request.

SU 4: The AA is gated to SAR and H. Operand 1 address is gated from K to L.

H (19, 20) are gated to AE (1, 2) and AEOB to ER for pack and unpack (Figure 6477). If the overlap triggers are not set, these bits are not used.

SU 5: The VFL fetch request trigger is set. The second operand byte address is gated from H (21-23) to the S latch and S is released. The entire second operand address is gated from H to incrementer to K, as in cycle three.

SU 6: The VFL address advance line is up during SU 6 in preparation for the addition of B2+D2+ (1 or 0) in SU 7. A one is forced to AA (28) if Z (1, 2, 4) is greater than S (1, 2, 4) and Z (8) is on. The sequence waits in SU 6 for an accept from the BCU.

This fetch request is actually initiating the first prefetch. The following table shows the number of storage words involved for the various length and starting byte address relationships.

Number of Words	Z (8)	Z (1,2,4) > S (1,2,4)
1	0	No
2	0	Yes
2	1	No
3	1	Yes

The starting address comparison is started in SU 6 by subtracting L from K and putting the result in K. This is the desired result for all instructions except unpack. In unpack, if this result is negative (no AM C Out 1), L is complemented through the main adder and put in K at the end of SU 7. This checks the magnitude of the difference since operand 1 can start to the right of operand 2 and move to the left during the execution (see Figure 42).

H (19, 20) are gated to AE (1, 2) and AEOB to SC in anticipation of overlapping fields.

The GT L with S trigger is set here for all instructions. The gate L with S trigger is reset as gate K with S trigger is set.

SU 7: If Z (1, 2, 4) > S (1, 2, 4) or Z (8) = 1, the VFL fetch request trigger is set and prefetch trigger 3 is set with SU L7. PF 3 is the accept wait cycle of the prefetch sequence and is followed by PF 4, which transfers J to M. L is gated to AM T/C and the complement trigger is set. For unpack, if there was a carry out of AM(1) at the end of SU 6, AOB is gated to K with SU L7. The AM carry out 1 trigger is blocked from changing with SU L7.

SU 8: The address comparison is completed in this cycle by setting 0-7 overlap or 8-15 overlap trigger if the conditions are met. The K zero detect generates two lines, K 0-27 equal zero and K 0-28 equal zero. Set 0-7 overlap trigger if K 0-28 equal zero. Set 8-15 overlap trigger if K 0-27 equal zero and not K 0-28 equal zero.

The GT TD out trigger is set during SU L8 for the instructions that use data from operand 1.

For pack and unpack, the ER and SC are gated to AE during SU 8 and SU 9. The AE complement trigger is also set for two cycles. This is done to check the ER/SC for equal. The gates are up for two cycles because the AE HS equal zero line has a long path to set the gate K with S trigger at SU L9.

Operand 1 is gated from J to K when J loaded is on and not single cycle mode. For single cycle, operand 1 was put in M during SU 2 and is gated from M to K during this cycle.

SU 9: Operand 2 is gated from J to L when J loaded trigger is on. SU L9 is enabled with the J loaded trigger on.

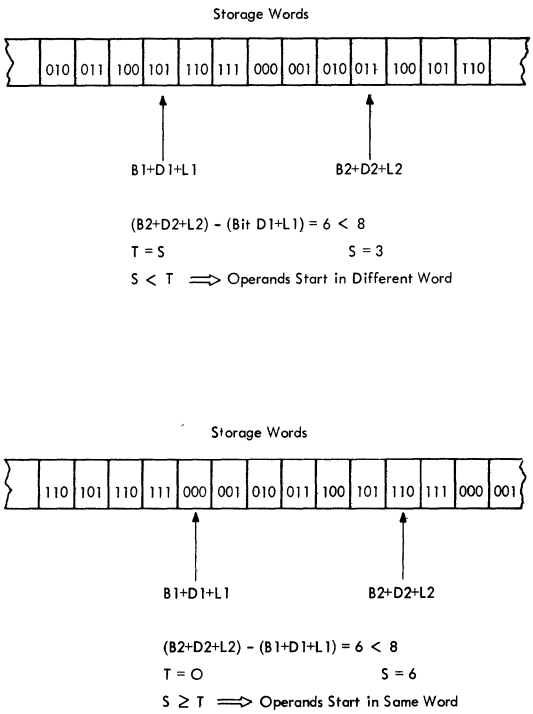
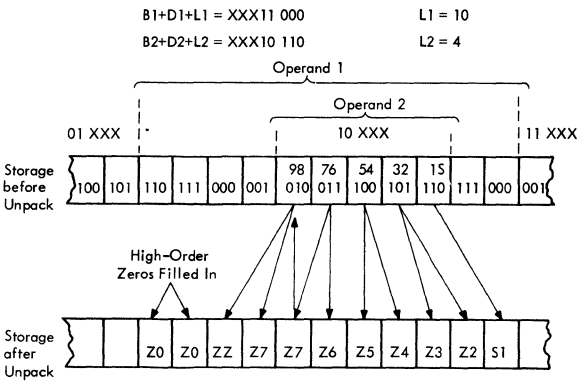


FIGURE 41. OVERLAP, BYTE ADDRESS RELATIONSHIP



This is an example of overlapping fields showing how the Op 2 word is modified during execution but prior to its use. Byte 2 (10 010) of Op 2 is changed from 98 to Z7 as a result of unpacking byte 3 (10 011).

FIGURE 42. UNPACK - OVERLAPPING FIELDS

The overlap triggers and AE HS are sampled for a set to GT K with S trigger at SU L9.

For single cycle mode, the prefetch storage request is delayed from SU 7 to SU 9. Delayed with the fetch request is the set of PF T3.

SU L9 sets the first iteration sequencer.

Set-Up Sequence -- Logical Instructions

- Fetch storage word that contains first byte to be processed for each operand.
- Compare storage addresses of each operand to determine if operands overlap in same storage words.
- Set starting byte address of each operand into S and T pointers.
- Start first prefetch if required.
- Set fetched words into K and L registers.
- Set VFL control and gating triggers.
- Set up the ER and SC to be used as a word count.

All of the VFL logical instructions process from low-order storage to high-order storage. IOP bits 8-15 specify an eight-bit length which applies to both operands.

Because many similarities exist between the set-up sequences for decimal instructions and for logical instructions, only the differences are explained below. Details are presented in set-up functions.

A word count is maintained in the ER, which is reset to zero during the set-up sequence. The ER is advanced by one each time a result word is stored. The increment gated to the AA for address generation comes from the SC. The amount in the SC is the increment needed for the next fetch, that is, if Op 2 is crossing word boundaries ahead of Op 1, the SC = ER + 2 for prefetch; if Op 1 is crossing word boundaries ahead of Op 2, the SC = ER + 1 for prefetch (see Figure 38). At the completion of prefetch, the SC = ER + 1 for the next Op 1 fetch.

A status trigger (VFL T5) is set during set-up if S is greater than T. T5 On indicates that Op 2 will cross word boundaries ahead of Op 1.

The first prefetch is initiated if Z (1, 2, 4) is greater than the complement of S (1, 2, 4) or Y (1, 2, 4, 8) not equal to zero or Z (8) is on. The Z-S comparison is made with complement S because the operands are processed from left to right (low-order to high-order storage).

The results of an edit or edit-and-mark instruction with overlapping fields are specified to be unpredictable. Therefore, these two instructions are always handled as though their operands do not overlap. Address comparisons are not made during edit and edit-and-mark set-up sequence.

Set-Up Functions

The following text explains the functions of set-up cycles, SU 1 through SU 9 for the VFL logical instructions.

SU 1: No increment is gated to AA since the desired starting address is B1 + D1.

SU 2: The VFL fetch request trigger is set with the B clock. The output of this trigger goes to the I unit to indicate J as the return address and to the BCU to set their fetch request. The set of both VFL request triggers (fetch request and store request) is latched to generate the gate of the AA to SAR and H.

The AEOB is gated to the ER as a means of resetting ER to zero with correct parity.

SU 3: No increment is gated to AA since the desired second operand starting address is B2 + D2. The VFL address advance line is up during cycle three so that B2 and D2 will be in IOP during cycle four. The low-order three bits of H are gated to the T latch and T is released with SU L3. This puts the starting byte address of operand one in the T pointer. H (0-23) is gated to the incrementer. The latched output of the incrementer and incrementer extender is gated to K0-31. Because nothing is gated into the incrementer extender, its output is zero with correct parity. The AOB (32-63) is gated to the K register (32-63) at the same time to put zeros with correct parity into the low-order half of K.

The sequence is held up here until an accept is received from BCU. If an immediate accept is received from the BCU, SU 3 takes only one cycle. This prevents a second request from being made in SU 5 without an accept from the first request.

SU 4: The AA is gated to SAR and H registers. Op 1 address is gated from K to L register.

One is forced to AE (7) and AEOB is gated to the SC. This sets the SC to 1 to provide the first address increment required for later fetches.

For edit and edit-and-mark instructions Y and Z counters are reset to zero; during this cycle, Y and Z are stepped up in these instructions.

SU 5: The VFL fetch request trigger is set and the second operand byte address is gated from H (21-23) to the S latch and S is released. The entire second operand address is gated from H register to the incrementer to K register.

SU 6: The VFL address advance line is up during SU 6 in preparation for the addition of B2 + D2+ 1 in SU 7; a one is forced to AA (28).

The sequence waits in SU 6 cycle for an accept from BCU.

The starting address comparison is started in SU 6 by subtracting L from K and gating the result back to K and L. This provides the magnitude of the difference of the starting addresses of the two operands.

The gate L with S trigger is set during this cycle in preparation to gate Op 2 bytes from the L register. If the operands overlap storage words and the gate K with S trigger is set later, then gate L with S trigger is reset.

SU 7: The add during SU 6 generated the result (B2 + D2) minus (B1 + D1); the desired difference to be checked for logical instructions is (B1 + D1) minus (B2 + D2) or the complement of (B2 + D2) minus (B1 + D1). Therefore, during SU 7 cycle for logical instructions, the contents of the L register are gated to AMTC and complemented through the main adder. AOB is then gated to the K register.

The first prefetch is started during SU 7 if Op 2 is in more than one storage word and not in single cycle mode. If Z (1, 2, 4) is greater than the complement of S (1, 2, 4) or if Z (8) is on or if Y (1, 2, 4, 8) is not equal to zero, the VFL fetch request and PF 3 triggers are set. If in single cycle mode, the first prefetch is started during SU 9 cycle.

SU 8: The address comparison is completed in this cycle and the overlap triggers are set. If K0-28 equals zero, the 8-15 overlap trigger is set.

The gate T decode out trigger is set during SU L8 for those instructions that use data from Op 1.

If VFL T5 trigger is on, indicating the first word boundary to be crossed is in Op 2, 1 is added to the SC. This puts 2 in the SC, the increment needed for the next prefetch (see Figure 38).

The Op 1 word is gated from J to K when the J loaded trigger is on and not single cycle mode. For single cycle mode, Op 1 was put in M register during SU 2 cycle and is gated from M to K register during this cycle.

SU 9: SU 9 is the last cycle of the set-up sequence for the VFL logical instructions. The first word of Op 2 is gated from J to L register when the J loaded trigger is on. SU L7 is enabled with the J loaded trigger on.

Except for ED, EDMK, TR, and TRT instructions, if the 0-7 overlap trigger is on and S is less than or

equal to T, the gate K with S trigger is set.

For ED and EDMK instructions, the first byte of Op 1 is put in DB/DC, where it is held throughout execution. This byte is the fill character. The length counters and pointers are not stepped because this character is examined as all other pattern characters are.

The gate L with S trigger is set with SU L9 for ED and EDMK. This set is delayed because there is no gate for the RBG to AOE and the AOE is used during SU 9 for ED and EDMK.

The VFL T2 trigger is set with SU L9 for MVC if T = 0 and S = 0 and YZ is greater than 7 and not overlap. When the VFL T2 trigger is on, it causes 64-bit words to move instead of 8-bit bytes. This type of move is called transmit mode. Transmit mode is entered on any MVC when both operands start on word boundaries and there is at least one 64-bit word to be moved. The byte mode is initiated to move any partial words on the end of Op 2.

For VFL logical instructions, SU L9 sets iteration sequencer 2 (IS 2) except for MVC in transmit mode and PF started during SU 7.

Set-Up Sequence -- TR and TRT

The translate instructions differ from other SS instructions in that the byte addresses move irregularly through a translation table in storage. Operand 1 is still processed sequentially starting with the low-order storage byte (B1 + D1). For this reason, source bytes are fetched one at a time from storage. Each operand 2 address that is formed is compared to the word address of the operand 1 word currently in the K register. If the table byte is in K, the gate K with S trigger is set and K is used for the source byte.

When the operand 1 address is formed, it is transferred to K. From K, bits (24-28) are gated to AOE and the AOE is gated back to K (24-31), thus setting K (29-31) to zero. K is transferred to M and subtracted from each operand 2 address. If any difference is within 0-7, the 0-7 overlap trigger is set and this causes the gate K with S trigger to be set. When operand 1 word boundaries are crossed, the Y-Z latch is added to B1 + D1 to generate the fetch address. Using this method of generation gives an address with the low-order three bits zero.

Set-Up Functions (See Figure 6482)

SU 1: The VFL fetch request trigger is set with SU T1 for single cycle operation. This early set is not required for translate but is used for simplicity since all other SS instructions advance the request for single cycle.

SU 2: Set VFL fetch request trigger and gate AA to SAR and H.

SU 3: Gate H (21-23) to the T pointer and H (0-23) to K.

SU 4: The low-order three bits of the address in K are set to zero by gating K (24-28) to AOE and AOE (0-7) back to K (24-31).

The Y and Z counters are reset to zero. For the translate instructions, Y and Z start at zero and are counted up until equal to IOP (8-15).

SU 5: The adjusted address in K is transferred to M. From there, it will be compared to each operand 2 address for possible overlap.

SU 6 and SU 7: SU 6 and SU 7 perform no function for the TR and TRT instructions.

SU 8: The first word to be translated is transferred from J to K. The gate TD out trigger is set with SU L8. This allows the byte from K, specified by T to pass through the LBG.

SU 9: The first iteration sequencer, PF T1, is set by SU L9.

Interrupts -- Set-Up Sequence

The only interrupt that can be initiated during set-up is address invalid. The address invalid trigger is normally set to the value of the address invalid line with each J advance. For SS instructions, the trigger can be set but not reset with J advance. With this arrangement, invalid address indications are accumulated and then the trigger is sampled at the end of set-up. The address invalid trigger is reset with SU 1.

The address invalid trigger in sampled at SU 9. If it is on, the sequence is switched to SEQ-T4 (SF 3) and the store-fetch trigger is set. These two, together, make SF 3 which is the start of the end sequence.

Iteration Sequences -- Decimal Instructions

- Iteration IS 1, IS 2, and IS 3 cycles perform VFL byte gating.
- IS cycles are used to execute SS decimal and logical instructions.

This section describes the iterations for all SS decimal instructions except multiply and divide. Multiply and divide are described separately. All iterations

start with the following initial conditions:

1. Word B1 + D1 + L1 in K
2. Word B2 + D2 + L2 in L
3. IOP (8-11) in Y
4. IOP (12-15) in Z
5. Starting byte address for Op 1 in T
6. Starting byte address for Op 2 in S
7. Either gate L with S or gate K with S trigger on, depending on the state of the 0-7 overlap trigger
8. Gate TD out trigger on for MVO, CP, AP, SP

The first iteration sequencer is set with SU 9 latch. The iteration cycles continue until a word boundary is encountered or the execution is complete.

Add -- Subtract

The only difference between AP and SP is the setting of the true/complement trigger, VFL T3. Figure 43 shows the general data path for AP and SP.

The first cycle (IS 3) is for examining the signs and setting the sign trigger. Since the pointers are not stepped during IS 1, the sign decoding does not need to be latched to set VFL T3 at IS 1 A-clock.

When doing a true add, the right side parity adjust correct is for the excess six gating into AV. The line called "GT HOD Decimal True to Parity Adjust" gates "HOD Equal 4/5" to the "exclusive OR" with "Invert Sign". The two phrases of the "exclusive OR" gate PL and not PL as the adjusted parity. This adjusts for three possible changes to the sign byte:

1. The incoming sign is degated and the machine preferred plus sign is forced at the RBG digit gates.
2. If the high-order digit is gated decimal true, a decimal digit 4 or 5 changes the parity.
3. If the LBG (Op 1) sign is negative, the low-order bit of the forced sign is inverted to make the result sign minus.

The operation is not complete until every byte in both operands has been examined. When operand 1 is exhausted, VFL T2 is set, gate TD out trigger is reset and parity is forced to the left side AV input. When operand 2 is exhausted, VFL T1 is set, gate K/L with S triggers are reset and parity is forced to the RBG. When both operands are exhausted, SF 1 is set instead of IS 2. During this store-fetch sequence, the last result word is stored and one of the following happens:

1. The operation is terminated if the result is correct as stored (Figure 6455).
2. The sign of the result is set plus for a negative zero result and the operation is terminated (Figure 6457).
3. The first word of operand 1 is fetched to start recomplementation if the result is in complement form.

Figure 6456 shows the recomplementing sequence. The S and T pointers contain the same byte address. The S pointer controls gating of K bytes to the true/complement input of AV. The T pointer controls putting the bytes back in K.

For a true add, the VFL adder carry trigger is not released after Op 1 is exhausted. A carry from the high-order byte of Op 1 is held in the carry trigger until both operands are exhausted, at which time VFL T6 is set. (VFL T6 is set with AV carry and true add and T1 and T2 and SF 1 latch). For both true and complement adds, the RBG is zero detected after the Op 1 is exhausted.

If a nonzero digit is detected, Op 2 has a greater length than Op 1. Therefore, VFL T6 trigger is set to signal an overflow condition. One exception exists, however, in which Op 2 may exceed the length of Op 1 without causing an overflow error. After the end of Op 1 (T2 On), the first Op 2 byte processed may contain digits 01 without causing an overflow error if the operation is complement add and a borrow occurs. VFL T8 trigger is set at the beginning of the first iteration cycle following the set of T2. The conditions T2 On and T8 Off define the first cycle after the end of Op 1 and block RBG bit 7 from entering RBG zero decode, thereby forcing RBG to equal zero if all other bits are zero. If a nonzero digit is detected, VFL T6 is set. The decimal overflow interrupt is set with (AP + SP + ZAP) and (VFL T6) and (PSW bit 37) and (VFL end sequence trigger on).

The condition register is set during the last store-fetch sequence with SF 4 latch.

Compare (CP)

Figure 44 shows the general data path for decimal compare. If the two operands have like signs, operand 2 is subtracted from operand 1 to determine which is the larger. If the two operands have unlike signs, operand 2 is added to operand 1 and the sum is zero detected. If the sum is nonzero, the positive operand is the larger. If the sum is zero, the two operands are both zero and, therefore, equal.

The execution is not complete until all bytes in both operands have been examined.

The VFL T3 trigger is used as a true/complement trigger for CP, AP, and SP. Therefore, VFL T3 controls the true/complement and parity adjust gates. VFL T3 is off for true add and on for complement add. When doing a true add, the right side parity must be adjusted for the excess-six gating into AV. When doing a complement add, the right side parity is adjusted for the sign removal only.

The condition register is set during the last store-fetch sequence with SF 4 latch.

Move With Offset (MVO)

Move with offset is a combination move and shift left 1 digit as the name implies. Figure 45 shows how this shifting is accomplished. The DB-DC is used as a buffer to hold the HOD of each byte until the next cycle when it is gated into AV as the LOD. The L1 length determines the end of the operation. If Op 2 is exhausted before Op 1, the gate K/L with S triggers are reset and parity is forced to the RBG. All other gates are unchanged. This fills out the remainder of the destination with high-order zeros. See Figure 6453.

Pack (PK)

Figure 46 shows the general data path and the sequencers used for pack. The DB-DC is used as an intermediate result buffer. DB/DC must be used because K is used as temporary storage and the result byte cannot be put in storage until it is complete.

One Op 1 byte is put in K at the end of IS 1 and every IS 3. This means that the store-fetch sequence is entered from IS 1 or IS 3 and always returns to IS 2. An Op 2 byte is used for each cycle, IS 1, IS 2, and IS 3. The prefetch sequence can be entered from any of the three sequencers. Therefore, the VFL T2 trigger is set with (S = 0) and (IS 2 latch) to remember which IS sequencer should be turned on after the prefetch. See Figure 6479.

The Op 1 length determines the end of the operation. If Op 2 is exhausted before Op 1, the gate K/L with S triggers are reset and parity is forced to the RBG. This fills the remaining Op 1 bytes with high-order zeros.

Unpack (UNPK)

Figure 47 shows the general data path and the sequencers used for unpack. The first cycle is the same for pack and unpack. For unpack, one Op 2 byte generates two Op 1 bytes. The Op 2 bytes must be fetched from K or L only once because the first Op 1 byte generated for a given Op 2 byte may be stored on top of the generating Op 2 byte (Figure 42). As an example, assume the two operands were located in the same storage word. Op 2 byte 3 could be generating unpacked Op 1 bytes 3 and 4. For this reason, the Op 2 bytes are put in DB/DC during IS 2 and DB/DC is used during IS 3. Here, as in pack, DB/DC is only needed for overlapping fields. Since it gives the correct result for nonoverlapping fields also, only one method of execution is used. (See Figure 6480)

The prefetch sequence is entered from IS 1 or IS 3. The store-fetch sequence can be entered from IS 2 or IS 3. Therefore, the VFL T2 trigger is set with (T = 0) and (IS 2 latch) to remember which sequence should be set after the store-fetch is completed.

PSW (12) determines which zone code is used for the unpacked result. PSW (12) equal to zero gives the BCD zone of 1111. PSW (12) equal to one gives the ASCII zone of 0101. The BCD zone is forced at the digit gates and PSW (12) controls the gating of AV bits 0 and 2 back to K. Removing bits 0 and 2 from the BCD zone gives the ASC zone but does not change the parity.

Zero and Add (ZAP)

Figure 48 shows the general data path and the sequencers used for zero and add. The first Op 2 word is set into L with SU 9 latch and, therefore, IS 3 is used to examine the Op 2 sign. The polarity of the sign must be known so that the machine preferred sign can be forced. (See Figure 6454)

When Op 2 is exhausted Op 1 is filled out with high-order zeros. When Op 1 is exhausted first, the remaining bytes of Op 2 are zero detected for overflow. VFL T6 trigger is set if an overflow condition occurs. VFL T1 and T2 triggers are set when Z and Y are counted down to 1110. These are used to generate gates for exhausted Op 2 and Op 1 conditions.

The condition register is set during the last store-fetch sequence with SF 4 latch.

Iteration Sequence -- Logical Instructions

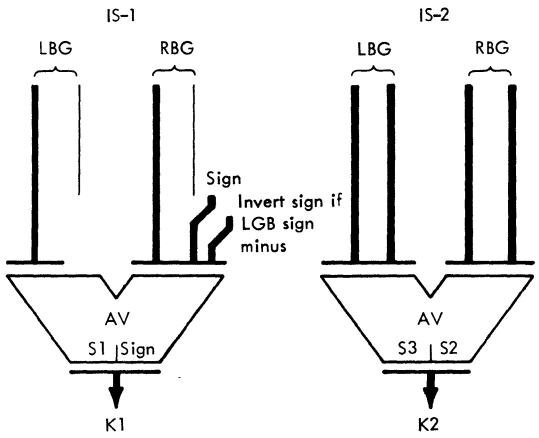
This section describes all logical SS instructions except the edits and translates. One iteration sequencer, IS 2, is used for all these instructions with the exception of move transmit mode. Move transmit mode used the store-fetch and prefetch sequences only. The iterations start with the following initial conditions:

1. Word B1 + D1 in K
2. Word B2 + D2 in L
3. IOP (8-15) in Y and Z
4. Starting byte address for Op 1 in T
5. Starting byte address for Op 2 in S
6. Either gate L with S or gate K with S triggers on depending on the state of the 0-7 overlap trigger
7. Gate TD out trigger is on for all instructions except MVC

The first iteration sequencer is set with SU 9 latch. The iterations continue until a word boundary is encountered or the execution is complete. The condition register (CR) is set for CLC, NC, OC, XC, TRT and EDMK during the last store-fetch sequence with SF 4 latch.

AND, OR and Exclusive OR (NC, OC and XC)

The SS logical connectives use the AOE. The output of the AOE is normally the OR of the two inputs. If



Right inputs to AV are complement if True Subtract and Decimal True if True Add.

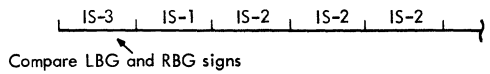


FIGURE 43. DECIMAL ADD OR SUBTRACT

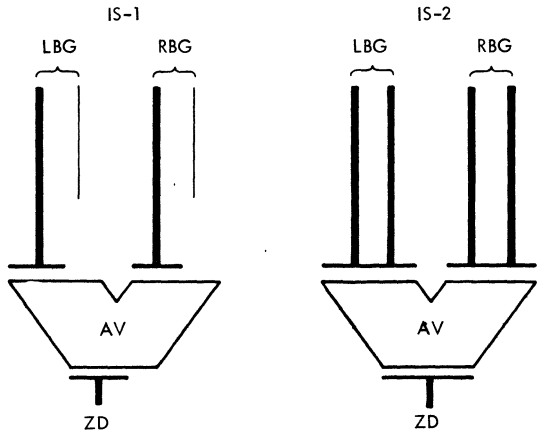


FIGURE 44. BASIC DATA FLOW - CP

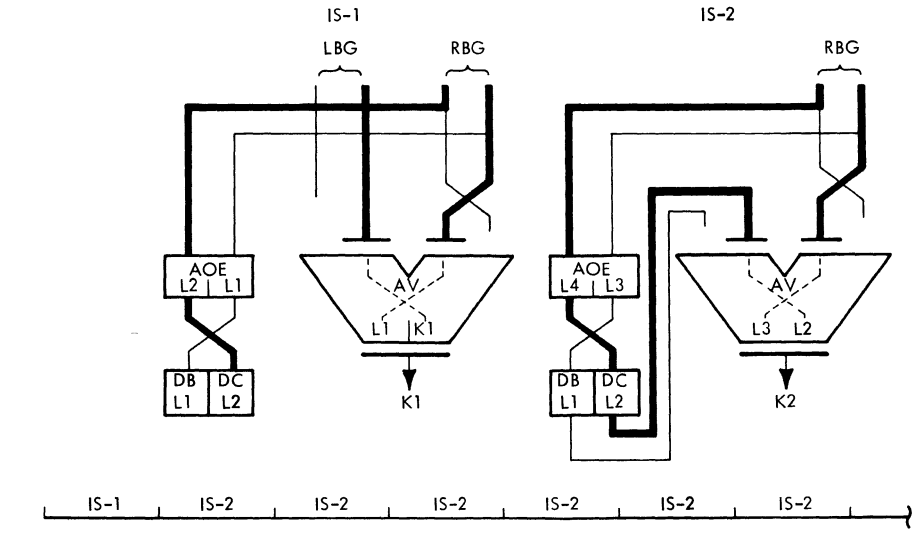


FIGURE 45. BASIC DATA FLOW - MVO

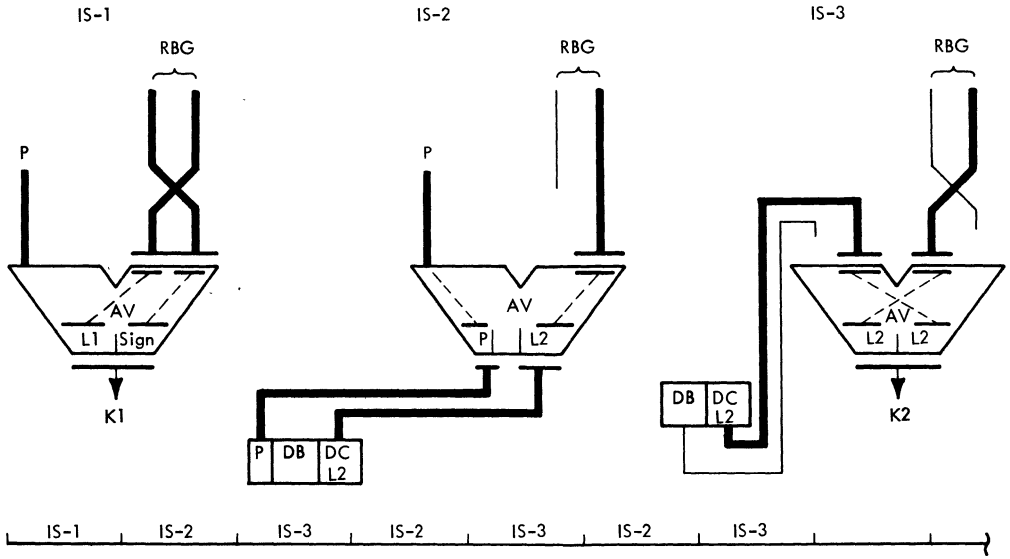


FIGURE 46. BASIC DATA FLOW - PACK

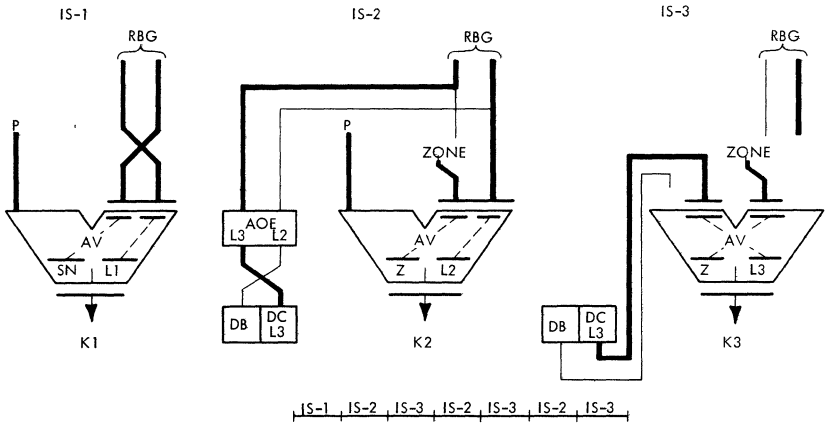


FIGURE 47. BASIC DATA FLOW - UNPACK

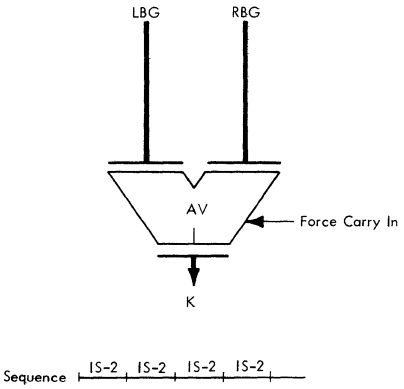


FIGURE 50. BASIC DATA FLOW - CLC

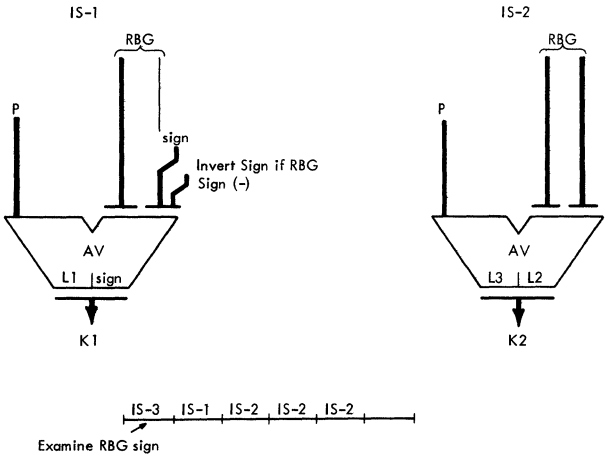


FIGURE 48. BASIC DATA FLOW - ZAP

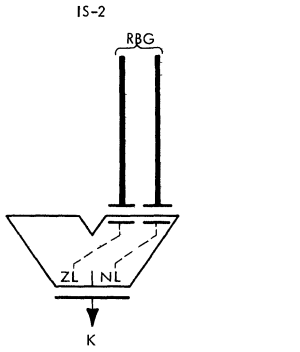


FIGURE 51. BASIC DATA FLOW - MVC

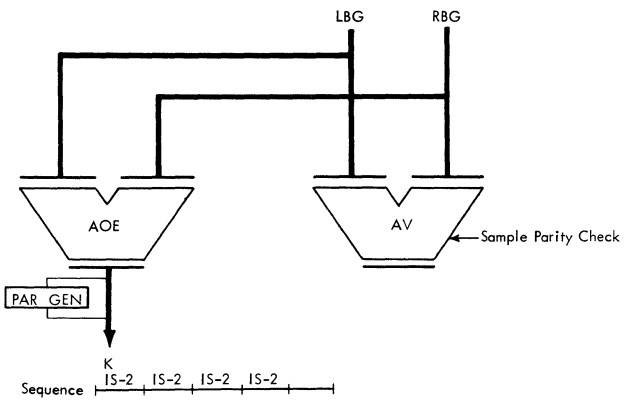


FIGURE 49. BASIC DATA FLOW, NC, OC, XC

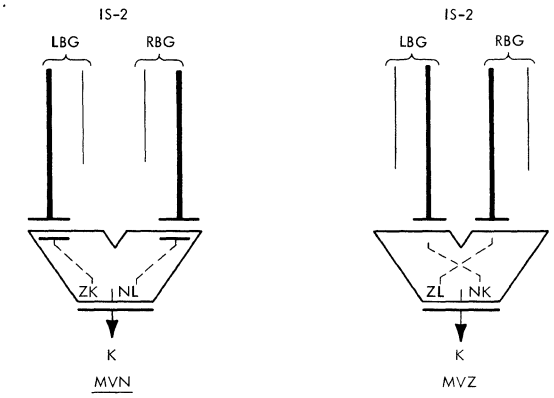


FIGURE 52. BASIC DATA FLOW - MVN, MVZ

either the AND or the Exclusive OR function is desired, a gating line must be activated. Figures 49 and 6464 show the data path and timing for NC, OC, and XC.

Parity is generated for the output of the AOE. The incoming parities are checked by gating the two bytes into the AV and checking the half sums.

Compare Logical (CLC)

The SS logical compare moves from left to right through the operands, making a byte by byte comparison. The operation continues until an unequal comparison is found (AV sum nonzero) or the operands are exhausted. Figures 50 and 6460 show the data path and timing for CLC.

Move (MVC)

Move (MVC) moves data from one location to another. Normally, execution is one byte at a time. However, if the two operands are not overlapped, start on word boundaries and are more than one word in length, the move is done one 64-bit word at a time. This is called move transmit mode. If the operands do not end on word boundaries, the transmit mode reverts back to the normal byte mode.

The first Op 1 word is fetched during set-up because it may be needed for execution of overlapping fields. After set-up, the Op 1 word is stored but not fetched. When the two operands start within eight bytes of each other, but in different storage words, the Op 2 bytes in the second word that do not actually overlap Op 1 must be fetched. This is done by fetching the first Op 1 word. Once Op 2 moves into the overlap area, no more storage words are required because the next Op 2 word is being generated in K.

Bytes are moved from L to K, or K to K depending on overlap conditions. Both HOD and LOD are gated binary on the AV right side and parity is forced to AV left. The AV output is put in K.

If both operands start on 64-bit word boundaries, at least two words apart in storage and there are more than eight bytes to be moved, the move transmit mode is entered. Move transmit mode moves 64-bit words. In the transmit mode, the first Op 2 word fetched is transferred to K and stored just as soon as an accept is received from the second Op 2 fetch. When each store is completed, another prefetch is started (Figure 9467). The first cycle gates M, the prefetched word, to AM T/C. The AOB is gated to L, and K. This is the next word to be stored. If another Op 2 word is to be fetched, the prefetch sequence continues after PF 1. If the word in K is the last full word to be moved, the store-fetch sequence follows PF 1. If a partial word remains, the move reverts back to the byte mode and the IS 2 sequence follows PF 1. The VFL T2 trigger is turned on with SU 9 latch if the conditions are met for transmit mode.

The Y-Z length counters are decremented by 8 with each SF 3 latch when in the transmit mode (T2 on). If an even number of words are to be moved, the low-order three bits of the length would be ones (Z = X111). Therefore, the Y-Z counters are equal to 1111-1111 when one word remains to be stored. This value in Y-Z causes SF 3 to follow PF 1 and the last full word is stored. If there are two odd bytes to be moved, in addition to the full words, the three low-order bits of the length (Z) would be 001. With this value in Y-Z, IS 2 follows PF 1. The partial word is moved one byte at a time (Figure 51) in order to set the mark register correctly. The sequence is switched from IS 2 to SF 3 when the YZ latch equals 1111-1110. This is the end of operation condition for the byte move.

Move Numerics and Move Zones (MVN and MVZ)

These two moves take a part of each Op 2 byte, either zone or numeric, and put it in a byte, leaving the remainder of the Op 1 byte unchanged. Figure 52 shows the data path and timing for the MVN and MVZ. For MVN, the LOD is gated to AV right and the HOD is gated to AV left. For MVZ, the HOD is gated to AV right and the LOD is gated to AV left. Both operands are fetched. The source is gated from L or K, depending on the overlap conditions.

Edit and Edit-and-Mark (ED and EDMK)

The initial conditions for the edits after set-up are:

1. Word B1 + D1 in K
2. Word B2 + D2 in L
3. Y and Z reset to zero
4. Starting byte address for Op 1 in T
5. Starting byte address for Op 2 in S
6. Gate L with S trigger on
7. Gate TD out trigger on
8. SC set to one
9. The first byte of Op 1 in DB/DC. This is the fill character.

The two sequencers used for edit iterations are IS 2 and IS 3. The IS 2 sequencer conditions the gates to unpack and validity check the HOD and sign detect the LOD of an Op 2 byte. The IS 3 sequencer conditions the gate to unpack the LOD of an Op 2 byte. One pattern byte is examined every cycle. Once a sequencer is turned on, it stays on until the conditions are met to unpack a digit. (These conditions being met are referred to as examine digit.)

Sequencer IS 2 (Figure 6471) is set with SU 9 latch. When the HOD is unpacked, IS 2 goes off and IS 3 is turned on. When the LOD is unpacked, IS 3 goes off and IS 2 is turned on. The S pointer is

stepped when going from IS 3 to IS 2. The iteration sequencers are on for an unpredictable number of cycles. The pattern bytes and S trigger (VFL T5) determine when a digit is unpacked. If during IS 2 cycle, the conditions are met to unpack a digit and the LOD is a sign (1010 - 1111), the S pointer is stepped and IS 2 repeats. Stepping S and not turning IS 3 on skips over the sign so it is not unpacked into the result. When a sign is detected, if it is positive (1010, 1110, or 1111), the S trigger is reset.

On each cycle that a source digit is not examined, either the fill character is gated from the DB/DC to K or K is left unchanged.

The zone that is forced, either 1111 for BCD mode or 0101 for ASCII mode, in unpacking digits depends on the PSW bit 12. The PSW bit 12 off (0) indicates BCD mode and the bit on (1) indicates ASCII mode. The BCD zone is forced at the digit gates, on the AV right side input. If the PSW bit 12 is on, bits 0 and 2 of the AV output latch are degated and do not go to K. This changes the BCD zone to an ASCII zone but does not change the parity since an even number of bits are removed.

The VFL status triggers are used for edits, as follows:

VFL T1: Remembers that the S pointer should be stepped when returning to the iterations from a store-fetch or mark sequence.

Set either IS 3 latch and examine digit latch or IS 2 latch and examine digit and edit sign latch.

Reset (set blocks reset) IS 2 latch.

VFL T2: Remembers which sequencer should be turned on when returning to the iterations from a store-fetch or mark sequence. On indicates IS 3, off indicates IS 2.

Set either IS 2 latch and examine digit and not edit sign latch or IS 3 latch and not examine digit.

Reset (set blocks reset) either IS 2 latch or IS 3 latch.

VFL T1 and VFL T2 combinations and sequences are:

IS 2 → SF + MARK → IS 2	Step S	T1	$\overline{T2}$
IS 2 → SF → IS 2	- -	$\overline{T1}$	$\overline{T2}$
IS 3 → SF + MARK → IS 2	Step S	T1	$\overline{T2}$
IS 3 → SF → IS 3	- -	$\overline{T1}$	T2

VFL T3: Remembers the zero field condition for a source RBG number. It is used to set the condition code.

VFL T3 is set by the SU 9 latch or the edit field Separation latch.

It is reset by the examine digit latch and not zero digit latch.

VFL T4: Remembers that a prefetch is required after a store-fetch. S pointer equal seven is not sufficient information to start a prefetch. Examine digit and edit sign latch indicate whether the last digit has been used. At the end of store-fetch this information is gone.

VFL T4 is set by the IS 3 latch and S = 7 and examine digit or edit sign latch.

It is reset by the PF 1 latch.

VFL T5: Used as the S trigger.

It is set by the digit select latch and not edit zero digit latch or significant start latch.

VFL T7: Holds address invalid indication for source field until a digit is used from the invalid word. VFL T7 and examine digit set the interrupt triggers.

VFL T7 is set by the address invalid latch and PF 4 latch.

It is reset by the ELC latch.

The following latches are used for edits to hold the control condition over a time:

Edit Digit Sel. LTH
Turned on by "LBG Equal Digit Select" LBG = 00100000

Edit Sign Start LTH
Turned on by "LBG Equal Sig. Start" LBG = 00100001

Edit Field Sep. LTH
Turned on by "LBG Equal Field Sep." LBG = 00100010.

Examine Digit LTH
Turned on by LBG equal to a Digit Select or Significance Start character.

Other Character LTH
Title refers to the off output. Turned on by LBG equal to Digit Select or Significance Start or Field Separator character.

Edit Zero Digit LTH
Turned on by Examine Digit and RBG LOD Equal Zero and (IS 2 or IS 3 trigger).

Edit Sign or RBG Not Zero LTH
Turned on by (RBG LOD Sign and Examine Digit and IS 2 trigger and ED + EDMK) or (AP + SP + ZAP + TRT and RBG Not Zero).

Edit Positive Sign LTH
Turned on by ED + EDMK and IS 2 trigger and RBG Sign Plus.

The HOD of each source byte is validy checked when it is examined. If it is invalid, the data check interrupt triggers are set if T7 is on. T7 on indicates that the word came from an invalid address and the invalid address interrupt is given priority.

Figure 6472 shows the mark sequence. The byte address is calculated by adding B1 + D1 + (Y-Z). The current pattern word is transferred to J while the address is inserted in GR1, then brought back to K during Seq D. The address is put in K (8-31) and then the high-order eight bits of GR1, which is brought out to the M register, are gated into K (0-7). K (0-31) is then put away in GR1. IS 1 returns the execution to:

1. IS 2 if not T2 and not set (SF 1 or SF 3)
2. IS 3 if T2 and not set (SF 1 or SF 3)
3. SF 1 if Y Z ≠ IOP (8-15) and T = 7
4. SF 3 if Y Z = IOP (8-15)

Translate and Translate-and-Test (TR and TRT)

The initial conditions for TR and TRT are:

1. Word B1 + D1 in K

2. Address B1 + D1 in M (low-order three bits zeroed)

3. Y and Z reset to zero

4. Starting byte address for Op 1 in T

5. Gate TD out trigger on

The TR and TRT are very much the same. Their differences are:

1. TR does not examine the translated bytes before storing them. TRT translates Op 1 bytes for examination and does not store any bytes.

2. TR is complete when all Op 1 bytes have been translated. TRT is complete when a nonzero byte is found or all bytes have been translated.

For translate, the address of the word in K (Op 1) is compared with each table address. If the difference is 0-7, the word being fetched is in K. In this case, the table byte is taken from K instead of the word returning from storage. Each time another Op 1 word is fetched, the low-order three bits of the address are set to zero and the address is put in M. Each table address is transferred from H to K and M is subtracted from K. The difference is put in K and K is detected for a value of 0-7. If K $\neq$ 0-7, gate L with S trigger is set. If K = 0-7, gate K with S trigger is set. While the address comparison is being made, the current Op 1 word is held in J.

Overlapping the table word return with the address calculation for the next fetch requires two Op 1 byte addresses. One is the byte address where the translated byte is to be stored (temporarily in K). The other byte address is that of the next byte to be translated. A hold is used on the T pointer latch to prevent it from changing after the T pointer register has been advanced. This holds the store byte address in the T latch to control in-gating of K. The T register is advanced to control the gating out of K. This gates the next Op 1 byte through the LBG to the AA for the address calculation. Figure 5038 shows a diagram of the T pointer and an example timing chart of T being counted.

Figure 6483 shows the arrangement of the sequencers to TR and TRT. Note that the end of the sequence, PF 1 and PF 2, is also the beginning of the sequence that follows.

VFL T8 trigger is used to prevent Y-Z from being stepped until after the first byte is translated. Y Z counter is stepped up until it equals IOP (8-15). Therefore, Y Z is stepped for each byte processed after the first byte. This corresponds to the definition of the operand length (i.e. the number of bytes to the right of the first byte).

After set-up and store-fetch sequences, VFL T1 trigger is off. It is set at PF 4 latch. VFL T1 trigger gates operations on data returned from storage. In the first sequence after set-up or store fetch, there are no words returning from storage (RBG).

For TRT, if a nonzero function byte is found, the mark sequence is started. The mark sequence starts with sequencer A. Figure 6484 shows the mark sequence for TRT. For TRT, the mark sequence ends the operation. The current byte count in Y Z is added to B1 + D1 to arrive at the byte address of the Op 1 byte which translated to a nonzero byte. This address is put in the low-order 24 bits of GR 1. This is accomplished by first putting the address in K and then gating the high-order eight bits of GR 1 to K. K0-31 are then put in GR 1. Then the contents of GR 2 are put in K and the nonzero table byte is inserted in K24-31. K0-31 is then put in GR 2.

Because a general register is being set during the last cycle, the VFL thru signal is delayed to IS 1 latch, just one cycle before the end. The VFL thru signal is then set to VFL end sequence trigger.

Prefetch Sequence

- Fetches Op 2 words from storage.
- Overlapped with iteration sequence.

The function of the prefetch sequence is to fetch Op 2 words from storage. While one Op 2 word is being processed, the next word is being fetched from storage and put in the M register. When an Op 2 word boundary is encountered, the prefetch sequence is initiated. The first cycle (PF 1) transfers M to L. After the first cycle, the iterations are started again. If another Op 2 word is required from storage, the remainder of the prefetch sequence (PF 2, 3, 4) is allowed to follow the first cycle.

For decimal instructions, the prefetch is initiated whenever the S pointer equals zero. For logical instructions, the prefetch is initiated whenever the S pointer equals seven. If the T pointer indicates an Op 1 word boundary has been reached at the same time as the Op 2 word boundary, the store-fetch sequence has priority over the prefetch.

The VFL address advance line is brought up with PF 1. The gate select register is set at B of PF 1 and the following A clock sets the right two-thirds of the SS instruction into IOP (16-31); this places the B2 and D2 address fields into IOP for use by the AA. The addressing adder sum at the end of PF 2 will be B2 + D2 + X where X is the VFL address increment.

First Prefetch: The first prefetch is started during set-up. Set-up sequencers control the address generation and set the VFL fetch request trigger. The second Op 2 word returns after set-up is completed and the iterations started. Therefore, PF 3 is set with SU 7 latch and reset with accept from the BCU. PF 4 is set with PF 3 latch and accept.

Since the first byte in an operand could be located anywhere within a storage word, there may be only one Op 2 byte to be processed before a second word is required. This means that the first prefetch, started during SU 7, would not be completed when the second prefetch is started. In this case, the PF 1 trigger is turned on but the latch is not enabled until the J loaded trigger is turned on. When PF 1 is turned on at the same time as PF 4, the M to AM T/C gating trigger is not set. With PF 4 on, J is being gated to AM T/C at this time and therefore J instead of M is gated to L with PF 1 latch.

Figure 53 shows three examples of the first source word boundary being encountered within one, two and three bytes of the start of execution. These examples assume that an immediate accept is received for the first prefetch request. Figure 54 shows an example where operand 2 has one byte in the first word and the accept to the first prefetch request is delayed two cycles.

Interaction with Store-Fetch

If both operands come to a word boundary at the same time (Figure 55, part A), the store-fetch sequence takes priority over the prefetch and is executed first. This is done for two reasons:

1. If the prefetch was allowed to go first, it would delay the store-fetch until an accept was received and possibly delay it even more waiting for the storage cycle to complete. With the prefetch following the store-fetch, most of it is overlapped with the iterations.

2. If the end-of-operation conditions exist, the prefetch sequence is not needed, and the store-fetch sequence ends the executions.

If an Op 1 word boundary is encountered or the end-of-operation conditions occur while a prefetch is in process, the start of the store-fetch is delayed until an accept is received for the prefetch request (PF 3 latch and accept).

Figure 55, part B shows the operands crossing word boundaries at the same time and Op 2 crossing a word boundary one cycle ahead of Op 1. In the latter of the two examples, two store-fetch sequences are shown for the two possible cases of storage interference. The first example shows the prefetch and store-fetch fetching from the same storage bank. The second example shows the prefetch fetching from the same storage bank that the store-fetch is storing in.

Note that with multiple storage units, the two requests made during a store-fetch sequence are always made to different storage units and therefore do not interfere with each other.

Decimal Instructions

The prefetch sequence is initiated if S = 0 and SF 1 or SF 3 is not being set.

The first Op 2 word, B2 + D2 + L2, is fetched during set-up. If a second word is required, the first prefetch is initiated during set-up and this fetches B2 + D2 + 0 or B2 + D2 + 8 bytes depending on whether the operand is in two or three storage words. When the first word boundary is crossed, if the Z counter is greater than 7, PF 2 is allowed to follow PF 1 and B2 + D2 + 0 is fetched. The field length limitation of 16 bytes limits the operand to a maximum of three storage words. If the operand is in two words, the prefetch initiated at the first word boundary consists of only one cycle, PF 1 to transfer M to L.

The S pointer is stepped even after the Z counter latch is equal to 1110 and the operand no longer enters into the result. It would be possible for a one cycle prefetch sequence to occur even though it is not needed. This is allowed because it should not happen very often; it only wastes one cycle out of many, and it is easier to prevent PF 2 from being turned on than it is to prevent PF 1 from being turned on.

Overlapping Fields: For AP, SP, ZAP and MVO, if the 0-7 overlap trigger is on, each prefetch moves Op 2 into the same storage word that Op 1 is in. For this reason, the gate K with S trigger is set at PF 1 latch and PF 2 is not enabled. If the 8-15 overlap trigger is on, M is transferred to L as usual. The word to be fetched is in K register and will be transferred to M on each store-fetch sequence, therefore no prefetch is initiated.

Pack and unpack must be handled differently for overlapping fields. In both instructions, the operands are used at different rates so that the initial address relationships do not hold throughout the execution. The overlap triggers are set according to the address comparisons:

Pack

$0 \leq (B2 + D2 + L2) - (B1 + D1 + L1) < 8$ sets 0-7 trigger

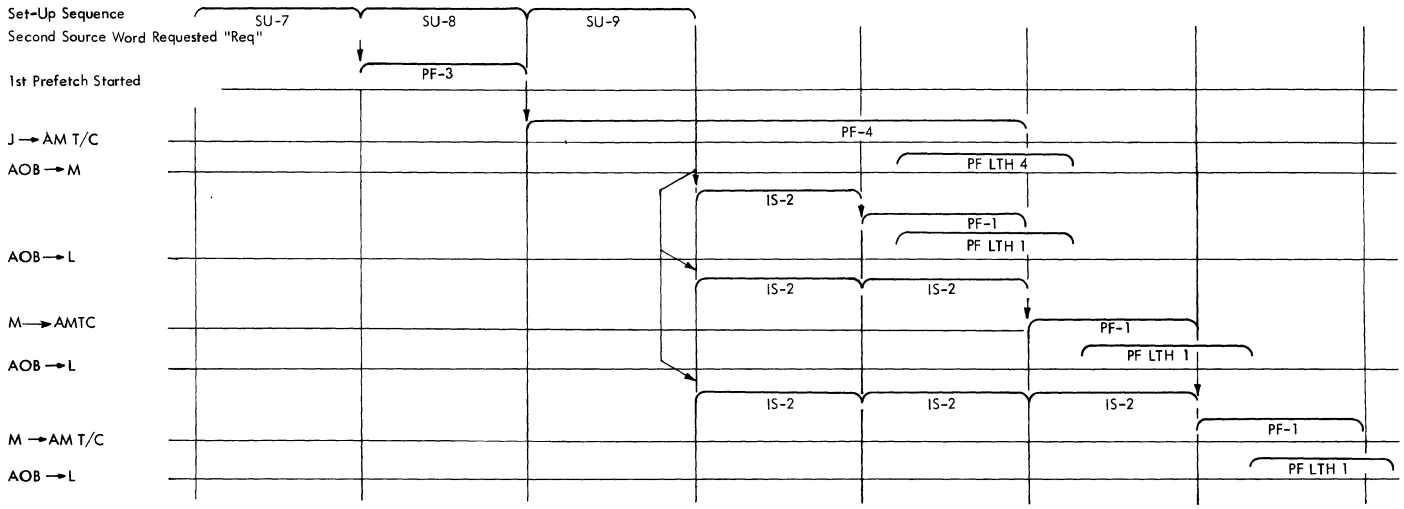
$8 \leq (B2 + D2 + L2) - (B1 + D1 + L1) < 16$ sets 8-15 trigger

Unpack

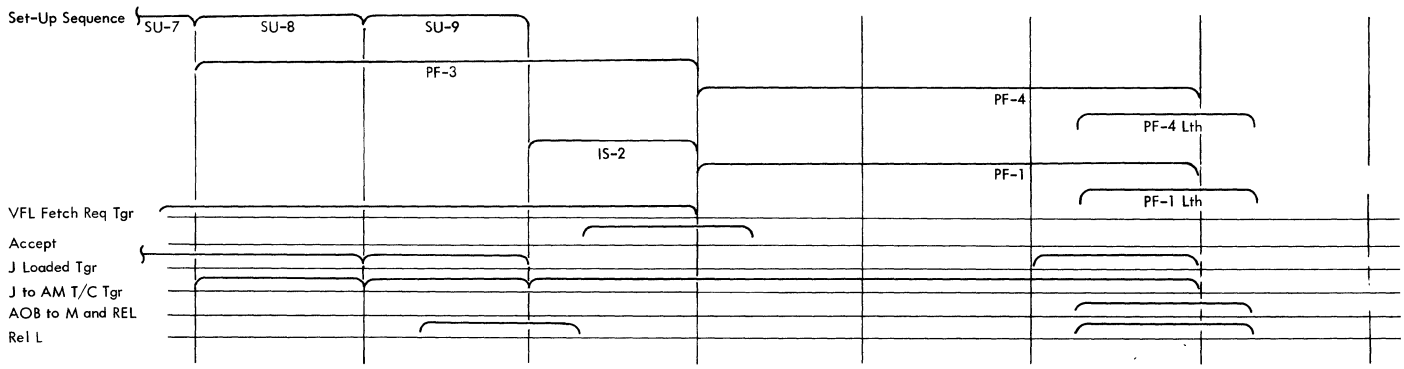
$-8 < (B2 + D2 + L2) - (B1 + D1 + L1) < 8$ sets 0-7 trigger

$8 \leq (B2 + D2 + L2) - (B1 + D1 + L1) < 16$ sets 8-15 trigger

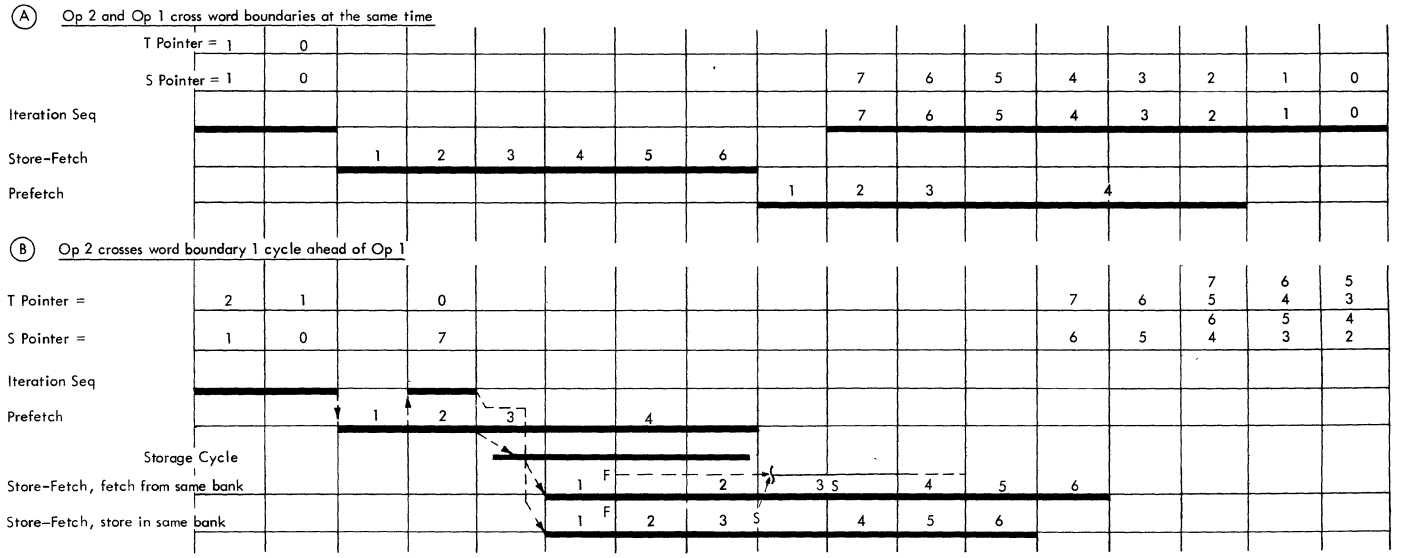
When either of these two triggers are on, the two low-order word address bits (H 19 and 20) are put in the ER and SC for Op 1 and Op 2, respectively. Each time a word boundary is crossed, the corresponding register is decreased by one and then the two register ER and SC, are compared for equal.



● FIGURE 53. FIRST PREFETCH



● FIGURE 54. FIRST PREFETCH - ACCEPT DELAYED



● FIGURE 55. PREFETCH/STORE-FETCH INTERACTION

If they are equal, the word boundary crossed moved the two operands into the same storage word. The address bits are in positions 1 and 2 of the ER and SC, and are decreased by adding 11 (the 2's complement of 01) to them. The SC is then subtracted from the ER in the AE. If all of the half sums of the exponent adder are equal to one, the two inputs are equal. Therefore, if AE HS = 1's, set gate K with S trigger, and if AE HS ≠ 1's, set gate L with S trigger.

The prefetch cannot be overlapped with the iterations when either of the overlap triggers is on. This is because the word being fetched might be in K register. The prefetch would fetch the word before the modified word (the result) had been stored. This would give an incorrect result. Therefore, Op 2 fetches are made after the Op 2 word boundary is encountered. The AOB is gated to L as well as M with PF 4 latch when one of the overlap triggers is on.

Logical Instructions

The two translate instructions do not use the normal prefetch sequence because Op 2 bytes are fetched one at a time from a table and do not follow in sequence.

The edit instructions do not overlap prefetch with iterations. The stepping of Y Z does not have a direct relationship to Op 2 bytes used and therefore, it is impossible to determine if another Op 2 word is needed until a word boundary is encountered. At an Op 2 word boundary, if Y Z ≠ IOP (8-15), a prefetch is started. If Y Z = IOP (8-15), the store-fetch sequence is started and, because it has priority, suppresses the prefetch sequence. It is still possible that an Op 2 word could be fetched that is not used; therefore, the address invalid latch and PF 4 latch set VFL T7. The first digit that is examined with VFL T7 on sets the invalid address interrupt.

Because edits use the two operands at different rates, two separate word counts must be maintained. The shift counter (SC) is used to hold a word count for operand 2. Each prefetch sequence adds one to the SC after it is used for the current fetch (the setup sequence sets the SC to one initially). The SC bits 2-7 are gated to the AA positions 23-28 to generate the address B2 + D2 + (SC x 8 bytes). The prefetch not being overlapped with the iterations means that:

1. The fetched word is put in the L register.
2. The iterations start after PF 4 instead of PF 1.

The other logical instructions overlap the prefetch with the iterations if there is no overlap, and do not fetch if either overlap trigger is on. For the nonoverlap condition, ER + 1 is put in the SC during PF 2 in preparation for the next store-fetch sequence. The word count of processed words is kept in the ER. The ER is incremented each time a result word is stored. Therefore, each store-fetch stores at B1 + D1 + ER, and fetches from B1 + D1 + (ER + 1). The store-fetch sequence leaves either ER + 1 or ER + 2 in the SC, depending on whether Op 1 is crossing word boundaries ahead of or behind Op 2. When entering either a prefetch or store-fetch, the SC contains the address increment to be used for the fetch. When iterations move through both operands at the same rate, the two operands cross word boundaries alternately.

For MVC, the store-fetch sequence starts at SF 3 and does not fetch. Therefore, the ER is transferred to the SC without incrementing it.

If the 0-7 overlap trigger is on, crossing an Op 2 word boundary moves Op 2 into the same storage word that Op 1 is in. Therefore, the gate K with S trigger is set to gate bytes of both operands from the K register.

Store-Fetch Sequence -- Decimal

- Fetches next Op 1 word from storage.
- Stores completed result word at Op 1 location in storage.
- Terminates instruction on last store.

The primary function of the store-fetch sequence is to store a completed result word and fetch the next Op 1 word to be processed. Six sequencers, SF 1 through SF 6, control the functions of each store-fetch sequence.

In general, whenever an Op 1 word boundary is encountered during iteration sequence, iterations are suspended and the store-fetch sequence is entered to store the completed result word and fetch the next Op 1 word to be processed. In this case, SF 1 cycle is the first of the store-fetch sequence; SF 1 and SF 2 cycles initiate the fetch request and then proceed through SF 3, SF 4, and SF 5, during which the store request is initiated. The store-fetch sequence then waits in SF 6 cycle for the next Op 1 word to arrive from storage into the J register. When the next Op 1 word arrives in J, the store-fetch sequence ends and iterations resume.

The store-fetch sequence may store only, fetch only, make two stores, or make no store or fetch. A store only sequence starts with SF 3. A fetch-store sequence and a fetch only sequence start with SF 1.

In all of these sequences, if the end execution conditions exist, VFL end sequence trigger is set with SF 3 latch and SF 5 is the last cycle. The SF 5 sequencer is the last cycle for all instructions with one exception, TRT. For TRT, if a nonzero function byte is found, the execution is completed with a

put-away sequence of A-B-C-D-IS1-IS3.

Figure 56 shows the various conditions that cause a store-fetch sequence and the sequencers used.

Store-Fetch for AP, SP (Figure 6455, 6456 and 6457)

There are two conditions that cause a store-fetch to be initiated:

1. T = 0, a word boundary is being crossed.
2. Y latch = 1110 and Z latch = 1110, both operands have been processed.

Both of these conditions start the store-fetch sequence at SF 1. The sequence has three different functions:

1. Crossing a word boundary or last store.
2. Change sign.
3. Start recomplement pass.

Crossing a Word Boundary or Last Store

This sequence is used for add/subtract first pass and the recomplement pass. When crossing a word boundary, a result word is stored and the next Op 1 word is fetched. When Y and Z equal 1110, the result has the correct sign and is in true form, the last result word is stored. The following is a description of the function of each cycle in this sequence:

SF 1: Gate eight to AA if two Op 1 words remain to be processed (Y > 7). Set the VFL fetch request trigger if another Op 1 word is required (Y ≠ 1110). This fetches (B1 + D1 + 8 bytes) if two words remain and (B1 + D1 + 0) if one word remains. If both operands have been processed (Y and Z = 1110) and the last word of Op 1 has not been stored (T5 on), the VFL store request trigger is set. This stores word (B1 + D1 + 0). The overlap triggers are reset with SF 2 latch, when Y-Z = 1110-1110, in preparation for recomplementing.

SF 2: Gate 16 to AA if two Op 1 words remain to be processed (Y > 7). Two words remaining mean the word just completed was the first one processed. Gate eight to AA if one Op 1 word remains to be processed (Y< 7). The sequence waits in SF 2 for an accept to come back from the BCU if a request was set during SF 1.

SF 3: T1 is set when Z = 1110 and T2 is set when Y = 1110. If L2 ≤ L1, the last store is made during SF 1 of the store-fetch sequence when Y and Z = 1110. When T1 and T2 are both on the store request is blocked at SF 3. If L2 > L1, the last store could be made during SF 3. This occurs when Y counts down to 1110 and an Op 1 word boundary is crossed before Z counts down to 1110. In this case, T5 blocks any further store request. If L2 > L1, and Z counts down to 1110 before an Op 1 word boundary is crossed, the last store request is made at SF 1.

The VFL end sequence trigger is set with SF 3 latch if T1 and T2 are on and the result is in true form or if the E interrupt trigger is on.

SF 4: The sequence waits in SF 4 for an accept from BCU if a request trigger was set during SF 3. if Y = 1110, SF 4 latch sets T5 to remember the last store has been made.

If the 0-7 overlap trigger is on, the gate L with S trigger is set. When the 0-7 overlap trigger is on it indicates that Op 2 and Op 1 were operating out of the same storage word. Op 1 crossing a word boundary moves that operand out of the storage word that Op 2 is currently in. During SF 5, the word in K is put in L where it continues to be used as a source word. The condition register is set with SF 4 if the VFL end sequence trigger is on.

SF 5: The state of the two overlap triggers indicates the difference between the starting addresses. If the 0-7 Overlap trigger is on, the two operands move in and out of common storage words. Each time an Op 2 word boundary is crossed, Op 2 is moving into the same storage word that Op 1 is currently operating out of. Each time an Op 1 word boundary is crossed, it is moving out of the word Op 2 is currently operating out of. If the 8-15 overlap trigger is on, the next word required by Op 2 is being generated as a result in K. Instead of prefetching, K is transferred to M as it is being stored.

If ELC was set with SF 4 latch, this is the last cycle.

SF 6: J is gated to K during SF 6. For one case, when the last result word is being stored and processing of Op 2 is not complete, no word has been fetched to J. However, J should be valid and it is gated to K.

If the address invalid trigger is on, SF 6 latch sets SF 3 instead of IS 2 and the operation is ended.

Change Sign

For the instructions AP and SP, if the result is zero it must have a positive sign. If the result is zero, it is in true form and does not require complementing. Therefore, there is never a need to change the result sign and prepare to recomplement during the same sequence.

If the result ZD trigger is off when Y and Z=1110 it indicates a zero result. The byte address of the sign byte (B1 + D1 + L1) is calculated. A byte with positive sign and zero digit is generated in the AV and placed in K at the address calculated, setting the mark corresponding to that byte. This byte is stored

Op Code Group	Conditions	SF 1	SF 2	SF 3	SEQA	SF 4	SF 5	SF 6
MVO	(T = 0) + (Y = 1110)			S		X	X	X
PACK, UNPK	[(T=0) + (Y=1110)] · (0-7+8-15 Overlap)			S		X	X	X
	[(T=0) + (Y=1110)] · (Not 0-7+8-15 Overlap)	F	X	S		X	X	X
ZAP	(T=0) · (Y ≠ 1110)			S		X	X	X
	(Y=1110) · (Correct Result Sign)	S	X	X		X	X	
	(Y=1110) · (Result is Neg Zero)	S	X	S		X	X	
CP	(T=0) · (Y ≠ 1110)	F	X	X		X	X	X
	(Y=1110) Set CR with SF-4 LTH	X	X	X		X	X	
AP,SP	(T=0) · (Y-Z ≠ 1110-1110)	F	X	S		X	X	X
	(Y-Z = 1110-1110) · (Result True) · (Result Sign Correct)	S	X	X		X	X	
	(Y-Z = 1110-1110) · (Result is Neg Zero)	S	X	X	S	X	X	
	(Y-Z = 1110-1110) · (Result Compl)	S	X	F		X	X	X
MVC	(T=7) + (Y-Z = 1111-1110)			S		X	X	X
MVN,MVZ,NC, DC, XC	(T=7) · (Y=Z ≠ 1111-1110)	F	X	S		X	X	X
	(Y-Z = 1111-1110)			S		X	X	
CLC	(T=7) · (Y-Z ≠ 1111-1110)	F	X	X		X	X	X
	(Y-Z = 1111-1110) Set CR with SF-4 LTH							
TR	(T=7) · (Y-Z ≠ IOP 8-15)	F	X	S		X	X	X
	(Y-Z = IOP 8-15)			S		X	X	
TRT	(T=7) · (Y-Z ≠ IOP 8-15) · (Function Byte = 0)	F	X	X		X	X	X
	(Y-Z = IOP 8-15) · (Function Byte = 0)			X		X	X	
ED,EDMK	(T=7) · (Y-Z ≠ IOP 8-15)	F	X	S		X	X	X
	(Y-Z = IOP 8-15)			S		X	X	

X Indicates sequencer is used.
F Indicates sequencer is used and VFL Fetch Req is set.
S Indicates sequencer is used and VFL Store Req is set.

The Y's and Z's used refer to the latched outputs.

FIGURE 56. STORE-FETCH CHART

and the operation ends with SF 5.

The following is a description, by cycle, of the change or invert sign sequence (see Figure 6457):

SF 1: With Y and Z=1110, there is no increment gated to AA. If a store is required, it is the (B1 + D1 + 0) word. If T5 is off, the VFL store request trigger is set.

In preparation for generating the starting byte address (B1 + D1 + L1), IOP (8-11) is gated to Y.

SF 2: The sequence waits here for an accept from the BCU if a request trigger was set during SF 1.

The gate of Y to AA (28-31) is started in SF 2 and continues through SF 3.

SF 3: The second request is normally set during this cycle; however, the BCU mark register must be set at the same time or before the BCU store request is set. To set the mark register, the address must be calculated, put in H, transferred to the T latch and then the mark register set. For this reason, the request is delayed one cycle. To maintain the normal ending sequence, the VFL Seq A is inserted between SF 3 and SF 4.

SEQ-A: The sign byte address is gated from H 21-23 to the T latch, which controls the K byte release and the setting of the mark register. The positive zero byte is generated and put in K. The VFL store request trigger is set but the normal function of gating AA to SAR and H is blocked. The address was set in SAR and H during the previous cycle.

The Seq A latch sets VFL end sequence trigger and generates a VFL through to the I-unit.

SF 4: Wait in SF 4 for the accept. The condition register is set with SF 4.

SF 5: Because the VFL end sequence trigger is set during SF 4 cycle, VFL end sequence latch gates the set of ELC trigger at the beginning of SF 5 cycle. Therefore, ELC and SF 5 terminate the instruction.

Start Recomplement Pass

Decimal data must always be in true form at the start and end of an operation. Therefore, if the result of an AP or SP is in complement form after the first pass, another pass must be made through operand 1 to recomplement it. Preparation for the recomplement pass is made during what would otherwise be the last store sequence.

A description by cycle of this sequence follows (see Figure 6456):

SF 1, SF 2: Same as change sign. The overlap triggers are reset with SF 1 latch so that none of the overlap functions are executed during the recomplementing pass.

SF 3: The VFL fetch request trigger is set to fetch (B1 + D1 + L1). This could be the word that was stored during SF 1 of this sequence, but to keep the controls as simple and straightforward as possible, the fetch request is always made.

SF 4: Wait in SF 4 for an accept for the fetch request made in SF 3. The starting byte address is gated from H to T latch to T register. To get T latch into T register unchanged, the count S and T down line must be degated with SF 4 latch.

T4 is set to remember that the following sequences are for recomplementing.

SF 5: The gate K with S trigger is set to gate K bytes through the RBG to the T/C + 6 gate.

T2 is reset since Y contains L1, and is no longer equal to 1110.

The IS 3 trigger is set with SF 5 latch to start the add/subtract sequence in the normal way.

SF 6: The SF 6 latch is enabled with the J loaded trigger. This means that the sequence waits here for the word requested at SF 4 to return. IS 3 latch is also enabled with J loaded trigger for AP or SP and T4.

Store-Fetch for ZAP, CP, MVO

This group of instructions has store-fetch functions similar to those of AP and SP but without all the variations.

Zero and Add--ZAP

The zero and add store-fetch sequence only stores. Therefore, if Y-Z $\neq$ 1110-1110 and T = 0, SF 3 is set and the sequence runs SF 3 through SF 6 (see Figure 6455). If Y-Z = 1110-1110, SF 1 is set and the sequence runs SF 1 through SF 5 (see Figure 6457). This last sequence stores a positive sign if the result was a negative zero. The details of these two sequences are described in AP-SP crossing word boundary or last store, and change sign.

Compare--CP

The decimal compare instruction does not store a result and therefore, the store-fetch is a fetch only sequence. The sequence starts at SF 1 and runs to

SF 5 or SF 6, depending on whether the operation is complete or not.

The following is a description, by cycle, of the CP store-fetch (see Figure 6455):

SF 1: Gate eight to AA if Y >7. Set the VFL fetch request trigger if Y ≠ 1110. The reset of the overlap triggers is for AP and SP in preparation for recomplementing

SF 2: Wait here for accept to fetch request if it was made.

SF 3: The VFL end sequence trigger is set if Y-Z = 1110-1110.

SF 4: Set T5 if Y = 1110. This blocks store-fetch from starting again until Y-Z = 1110-1110.

SF 5: Gate K to L for 0-7 overlap and K to M for 8-15 overlap (see AP, SP "Crossing a Word Boundary or Last Store" for more details).

SF 6: Gate J to K when J is loaded.

Move-with-Offset--MVO

Because move-with-offset (Figure 6455) is a move type instruction, it does not fetch Op 1. For storing only, the store-fetch routine is started at SF 3 and runs to SF 5 or SF 6. The SF 6 sequencer is used to separate SF 5, the last cycle sequencer, and the iterations. The SF 6 is normally used to gate J to K but no fetch is made for MVO.

T5 need not be set since VFL end sequence is set when Y = 1110.

Store-Fetch for PACK, UNPK

The PACK and UNPK instructions do not move through both operands at the same rate. This means that the starting address relationships do not remain static throughout the execution. Therefore, overlapping fields must be handled differently from other instructions.

Nonoverlapping Fields: If it is determined during set-up that the starting addresses are not close enough together to have overlapping fields, PACK and UNPK are treated like MVO. The store-fetch sequence is entered at SF 3 and the complete result word is stored. The VFL end sequence trigger is set when Y = 1110.

Overlapping Fields: The overlap triggers are set during set-up as follows:

0-7 if $0 \leq (B2 + D2 + L2) - (B1 + D1 + L1) < 8$ for Pack
or $-8 < (B2 + D2 + L2) - (B1 + D1 + L1) < 8$ for Unpk
8-15 if $8 \leq (B2 + D2 + L2) - (B1 + D1 + L1) < 16$ for Pack and Unpk

With these initial conditions and the field length limitations, it is possible to monitor the two low-order word-address-bits (H19 and H20) to determine when the operands are in the same storage word. These two bits are put in the ER and SC, positions 1 and 2, during set-up. Each time a word boundary is crossed, the corresponding address (ER for Op 1 and SC for Op 2) is decreased by one and then the two registers are compared. If the two registers are equal, the operands will be working on the same storage word.

Other details are similar to the AP, SP store-fetch sequence (see Figure 6455).

Store-Fetch Sequence--Logical

The main difference between the logical and decimal SS instructions store-fetch sequences is the address generation. A word count is maintained in the ER of the words processed. This word count can be used to generate the increments added to the base address for storing and fetching. This word count is advanced each time a result word is stored. The fetch preceding the store uses ER + 1 for the address increment. If the Op 2 field is crossing word boundaries ahead of Op 1, the prefetch address increment is ER + 2. If Op 1 is crossing word boundaries ahead of Op 2, the prefetch address increment is ER + 1. Since both operands move through storage at the same rate, a comparison of their starting byte addresses indicates which operand is leading throughout the entire execution. T5 is set during set-up of S > T, indicating operand 2 will cross word boundaries ahead of operand 1. At the end of either a store-fetch, or a prefetch, the SC contains the increment for the next fetch. Instructions MVN, MVC, MVZ, NC, CLC, OC and XC are included in this group.

MVC does not fetch Op 1. Therefore, the store-fetch sequence is started at SF 3 and the prefetch leaves the contents of the ER in the SC for the next address increment.

Following is a description of the unique operation of this store-fetch (Figure 6465):

SF 1: Transfer the ER to SC in preparation for storing result.

SF 3: Add one to ER for storage word count advance.

SF 5: Add one to ER and put sum in SC. This is prefetch address increment if Op 1 is leading Op 2.

SF 6: If T5 and neither overlap trigger is on, add one to the SC (ER + 2) and put sum in SC. This is prefetch address increment if Op 2 is leading Op 1. If either one of the overlap triggers is on, the prefetch is not overlapped with the iterations. When an Op 2 word boundary is crossed, the iterations are suspended while the next Op 2 word is fetched. For this case, the address increment is ER + 1.

Store-Fetch for ED, EDMK, TR, and TRT

For EDMK and TRT, it is necessary to calculate and put-away in GR 1 the full 24-bit operand 1 address. The easiest way of calculating this address is to count Y-Z up (starting with Y-Z = 0) instead of down and add Y-Z to B1 + D1, when the current byte address is needed. Y-Z register and Y-Z latch can then be used for addressing increments when storing and fetching operand 1 words at word boundaries.

ED and EDMK

For ED and EDMK (Figure 6473), the two operands move through their storage fields at different rates. The stepping of Y-Z corresponds to the processing of operand 1 bytes. The Y-Z counter has no direct relationship to operand 2. Therefore, a word count is maintained in the SC for operand 2.

TR and TRT

For TR and TRT (Figure 6485), the Op 2 storage references move at random through a translation table that can vary in size. The actual size of the table is not specified in the operation code. Therefore, it is impossible to make an initial address comparison to determine if there is possible overlapping of the fields. For this reason, the word address (three low-order bits equal zero) of the word in K is placed in M. Each table address calculated is compared to M and the difference is checked. If the difference between the table address and address of the current word in K is 0-7, the table byte required is in K. The gate K with S trigger is set and the word returned from storage is not used.

The address calculated during store-fetch for the fetch is the address to be put in M. This address must be transferred to M before the store address is put in H. Therefore, the word in K is transferred to M and the address in H is transferred to K. Then K and M are swapped to put the address in M and the result word being stored back in K.

DECIMAL DIVISION

Method of Division

To show the method of decimal division in the System/360 Model 75, consider first the normal long-hand method as shown below:

```
      176        R = 12
23 ) 4060
     -23
     176
     161
     150
     138
      12
```

To generate the quotient, the divisor is subtracted from the high-order end of the dividend as many times as possible. The number of times it can be subtracted is the value of the first quotient digit. The divisor is then shifted right one digit. The second quotient digit is developed by subtracting from this position. This process continues until the divisor has been shifted to the low-order end of the dividend and the last quotient digit is generated. This process is shown more clearly below:

```
                  176        R = 12
              23 ) 4060
         1        -23
                  1760
          1       -23
                  1530
          2       -23
                  1300
          3       -23
                  1070
          4       -23
                  840
          5       -23
                  610
          6       -23
                  380
          7       -23
                  150
            1     -23
                  127
            2     -23
                  104
            3     -23
                  81
            4     -23
                  58
            5     -23
                  35
            6     -23
Quotient  ------------------------  Remainder
          1 7 6             12
```

For each quotient digit generated above, the divisor is subtracted until the remainder is less than the divisor. Because it is difficult for the computer

to determine when the dividend is less than the divisor, the divisor is subtracted until the dividend goes negative. When the dividend goes negative, the divisor has been subtracted one time too many; therefore, the divisor must be added back to restore the dividend to a true value. The divisor is then shifted right one digit and the next quotient digit generated in the same manner. This process continues until the dividend is reduced to a value less than the divisor; this then, is the remainder.

Restoring Division (Figure 57)

Subtraction of decimal numbers in System/360 Model 75 is accomplished by the 9's complement method. In decimal division, the divisor is complemented and added to the dividend the required number of times to cause the dividend to go negative. A counter counts the number of times the divisor is subtracted from the dividend without causing the dividend to go negative. When the dividend goes negative the counter contains the value of the quotient digit for that decimal position. The dividend is then restored to true value by adding the divisor back. The digit value in the quotient counter is stored and the counter reset to zero.

The divisor is then shifted right one digit and subtraction repeated until the dividend again goes negative, and a new quotient digit generated. The dividend is again restored to a true value, the quotient digit stored and the process repeated until the value of the dividend becomes less than that of the divisor. Division is then complete.

Restoring division restores the dividend to a true value each time it goes negative.

Non-Restoring Division (Figure 58)

In non-restoring division, instead of adding the divisor back to restore the dividend when it goes negative, the divisor is shifted right one digit. The next quotient digit is then generated by adding the divisor to the dividend the number of times required to cause the dividend to go positive.

Non-restoring division generates quotient digits two ways:

1. When the dividend is positive, the quotient digit counter is set to zero and stepped up one each time the divisor is subtracted until the dividend goes negative.
2. When the dividend is negative, the quotient digit counter is set to 9 and counted down one each time the divisor is added until the dividend goes positive.

Combined Restore and Non-Restore (Figure 59)

The speed of the decimal divide process can be optimized by combining features of the restore and the non-restore methods. A review of the restore and non-restore method indicates that a quotient digit less than 5 can be generated faster by subtracting whereas, a quotient digit greater than 5 can be generated faster by adding.

In the System/360 Model 75, both the restoring and the non-restoring methods are used. The high-order digits of the divisor and dividend are decoded to predict approximately what the next quotient will be. This quotient prediction allows the selection of the method that will be the fastest for each particular quotient digit.

In general, the decimal division process is as follows: A divide check is made. The divisor is left-aligned with the left-most-but-one dividend digit. A trial subtraction (divide test) is made and if the result does not go negative the quotient and remainder will not fit into the Op 1 field. In this case, the divide check trigger is set and the division process is terminated.

If the result of the trial subtraction (divide test) is negative the operation continues. The divisor is then shifted right one digit and subtracted from or added to the dividend. Whether a quotient digit is generated by addition or subtraction is determined by the positive or negative state of the dividend and the predicted quotient. The quotient digit is generated in the digit counter (DC). When the first digit is completed, it is temporarily stored in the digit buffer (DB). The divisor is then shifted right 4 and the next quotient digit is generated. Now that a full byte of quotient has been generated it is put away in the upper end of the dividend-quotient field. This process continues until the divisor is right-aligned with the low-order byte of the dividend. The last quotient digit is then generated and put away. The division is then terminated.

Unit Functions

Execution of the decimal divide instruction utilizes the following registers and counters as temporary data storage and to provide controls during the various sequences involved.

Registers

The J, K, L, and M registers are used as temporary storage for the divisor and dividend words during the execution of the decimal divide instructions.

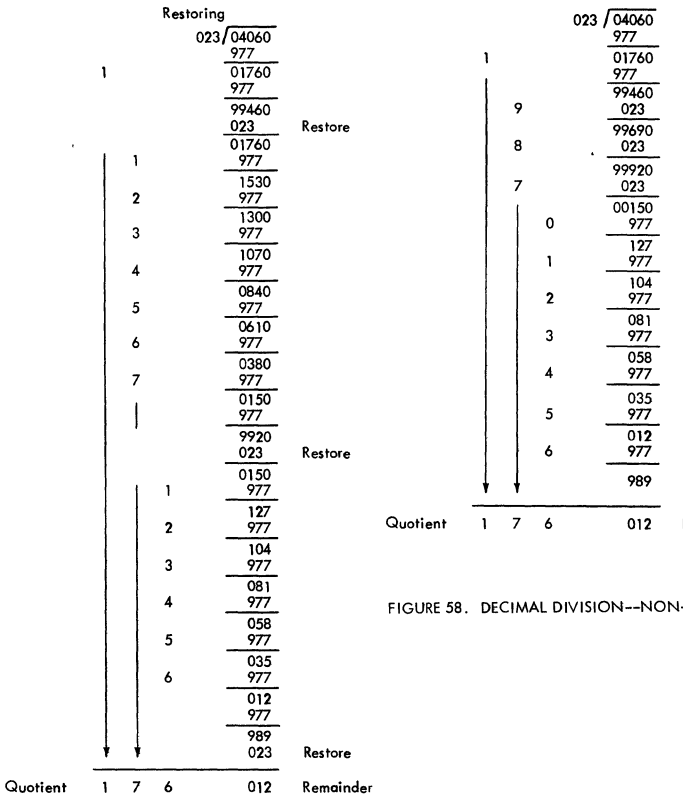


FIGURE 57. DECIMAL DIVISION--RESTORING

FIGURE 58. DECIMAL DIVISION--NON-RESTORING

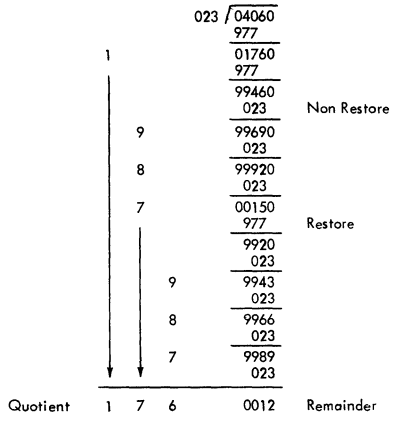


FIGURE 59. DECIMAL DIVIDE--COMBINATION RESTORE AND NON-RESTORE

J Register: The J register receives each word of the divisor and dividend when they are brough from storage during the set-up sequence. During divide iterations, the J register serves no function other than to retain the low-order dividend word when the dividend is in three storage words.

K and M Registers: The K register holds the portion of the quotient-remainder field that is presently being worked on. The M register holds the portion of the dividend that is on the other side of the word boundary if a word boundary is crossed by the present alignment of the divisor.

Because the quotient-remainder field can have a maximum length of 16 bytes, it can cross two word boundaries. The length of the divisor determines how much of this field is used in determining any one quotient digit. Because the maximum divisor length is 8 bytes, only one word boundary can be crossed by the portion of the dividend field being worked on.

At the beginning of the divide iteration, the high-order two words of the dividend field are in K and M. The first word that will be worked on is in K.

L Register: The L register holds the entire right-aligned divisor. Right-alignment is done during set-up. As each quotient digit is completed, the divisor is shifted right or left one decimal digit during sequence D cycles to provide the correct dividend-divisor alignment to generate the next quotient digit.

Counter Functions

Y Counter: Y is initially set to L2. Every time a byte of the dividend has been exhausted (a quotient byte generated) Y is stepped up by 1. When Y = L1 the last quotient digit is complete and the operation can be terminated.

In addition, Y counter provides the address increments when a new T pointer value is needed after each quotient byte is complete.

Z Counter: Z is set with L2 at the beginning of each pass (Seq A) through the divisor. It is stepped down by 1 every time a byte of the divisor in L is used. When Z latch = 1110, the addition has been completed with the exception of the extra byte during odd cycles.

S Pointer: S is reset (Seq A) at the beginning of each pass through the divisor. It is used to select the divisor bytes as they are subtracted from the dividend. S is stepped down by 1 as each divisor byte is processed.

T Pointer: T is set with B1 + D1 + Y at the beginning of each pass (Seq A) through the divisor. T is counted down by 1 as each byte of the dividend is processed.

Y is stepped up by 1 as each quotient byte is generated. Therefore, B1 + D1 + Y provides the T starting point that shifts right while proceeding through the division.

T also selects the K byte in which to set the quotient. At the end of a pass through the divisor when a quotient byte has been completed, it is only necessary to step T down once more (Seq A) to set the quotient byte into K.

Digit Counter: DC is used to generate the quotient digit. It is set to zero and counted up when the divisor is being subtracted from the dividend. It is set to 9 and counted down when the divisor is being added. It is stepped after every pass through the divisor until the dividend changes sign. When this happens, the quotient digit is complete.

Digit Buffer: DB is used to hold one quotient digit while another is being generated in DC. Thus, a full byte of quotient can be stored after every other digit is generated.

Shift Counter: The shift counter (SC) is used to gate odd-even cycles. During set-up, the SC is set to zero. Thereafter, when each quotient digit is complete, the SC is stepped up 1. When the SC contains an odd number, the pass through the divisor is an odd pass; when the SC is even, the pass through the divisor is even.

Odd-even passes are used to maintain correct divisor and dividend alignment. After each quotient digit is complete, the divisor is shifted right 4 bits (one decimal digit) in relation to the dividend. To effectively shift the divisor right 4, the first time it is only necessary to shift the contents of L register right 4. For the next right 4 shift, the starting point of the dividend is shifted right one byte (8 bits) and the divisor (L register) left 4 bits.

To keep track of this shifting, the SC is incremented after every quotient digit is generated. The odd cycles are defined as those during which the divisor is in its left-most position in L. During even cycles, the divisor is right aligned in L.

An odd cycle quotient digit is put away in DB. When an even cycle quotient digit has been generated in DC, DB and DC are put away in K.

Control Triggers

T1 -- First Word Store Trigger: T1 trigger is set during the SF 3 cycle of the first SF sequence. T1 is used in combination with the T4 to generate storage addresses for subsequent SF sequences (Figure 60). When set, T1 stays on until ELC.

T2 -- Restore Trigger: The T2 trigger is set to restore the dividend after each quotient digit is complete if the non-restore trigger is off. T2 is set during the sequence A cycle in which the quotient digit is complete, except the sequence A cycle of divide test or the end of a restore pass. When T2 is on a restore pass is forced; the divisor is added or subtracted from the dividend, depending on the status of T/C control T3 trigger. If the preceding pass through the divisor was a subtract pass and the quotient digit is complete, the dividend is negative when Seq A is entered. T2 trigger is set, if the non-restore trigger is off, and a restore pass adds the divisor back to the dividend to restore it to a positive value. The same occurs if the previous pass through the divisor was an add pass. When Seq A is entered and the quotient digit is complete, T2 trigger is set, if the non-restore trigger is off, and a restore pass subtracts the divisor from the dividend to cause it to go negative. T2 trigger is reset during the Seq A cycle that follows the restore pass.

T3 -- True/Complement Trigger: During decimal divide iterations, the T3 trigger controls whether the divisor is added to or subtracted from the dividend. If T3 is on it causes the divisor byte to be complemented at the TC +6 input gate to the decimal adder.

T3 is set during PF4 cycle of the decimal divide set-up sequence, and remains on during the divide test pass. Thereafter, T3 is set and reset during Seq A cycle when quotient digit is complete.

T4 -- Swap Trigger: The T4 trigger controls the swapping of K and M registers after each pass when a dividend word boundary is crossed.

T4 trigger is set during IS 1 or IS 2 cycles of the divide test pass if a dividend word boundary is crossed. T4 is also set during Seq A of a restore pass that precedes the crossing of a dividend word boundary.

T4 is reset during SF5 if the portion of the dividend being processed moves into one word.

T5 -- Block-Swap Trigger: When generating the last quotient digit of a word, the swap trigger, T4, is still on. However, no word boundaries are actually crossed during these passes through the

Divide Request Address	Fetch Req Cycle	Conditions	Word Returns From Storage	
			Cycle	To Register
$B_1 + D_1 + L_1$	SU 2	————	SU 7	K and M
$B_2 + D_2 + L_2$	SU 4	————	SU 9	L
$B_2 + D_2$	SU 7	S < Z (Sets T4)	SU 11	L
$B_1 + D_1$	SU 9	T4 · (T1 + T2)	PF 2	M
$B_1 + D_1$	SU 11	T4 · (T1 + T2)	PF 2	M
$B_1 + D_1 + 1$	PF 2	T1 and T2	PF 3	K

Dividend Field - Word Addresses

1 Word	$B_1 + D_1 + L_1$		
2 Words	$B_1 + D_1$	$B_1 + D_1 + L_1$	
3 Words	$B_1 + D_1$	$B_1 + D_1 + 1$	$B_1 + D_1 + L_1$

Divisor Field - Word Addresses

1 Word	$B_2 + D_2 + L_2$	
2 Words	$B_2 + D_2$	$B_2 + D_2 + L_2$

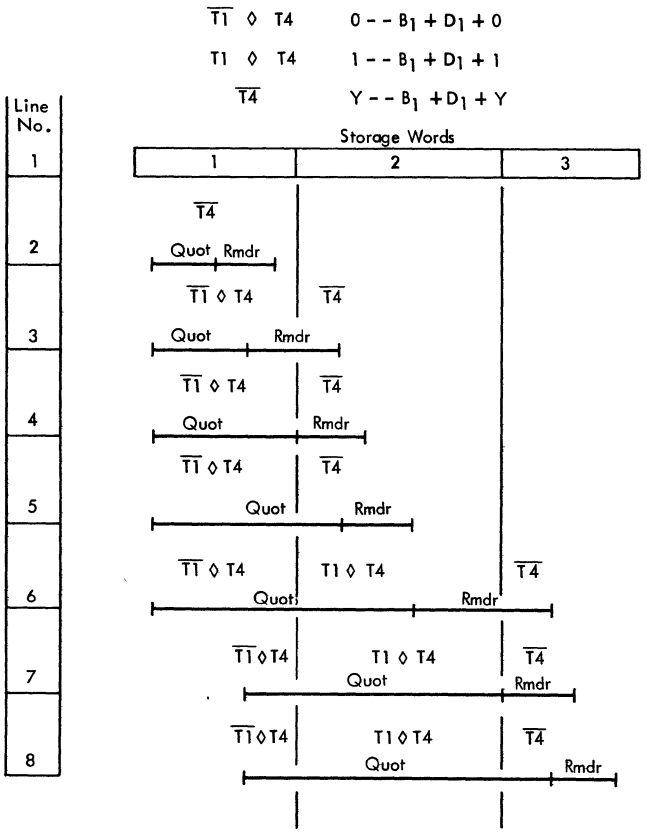


FIGURE 60. ADDRESSING--DP STORE-FETCH

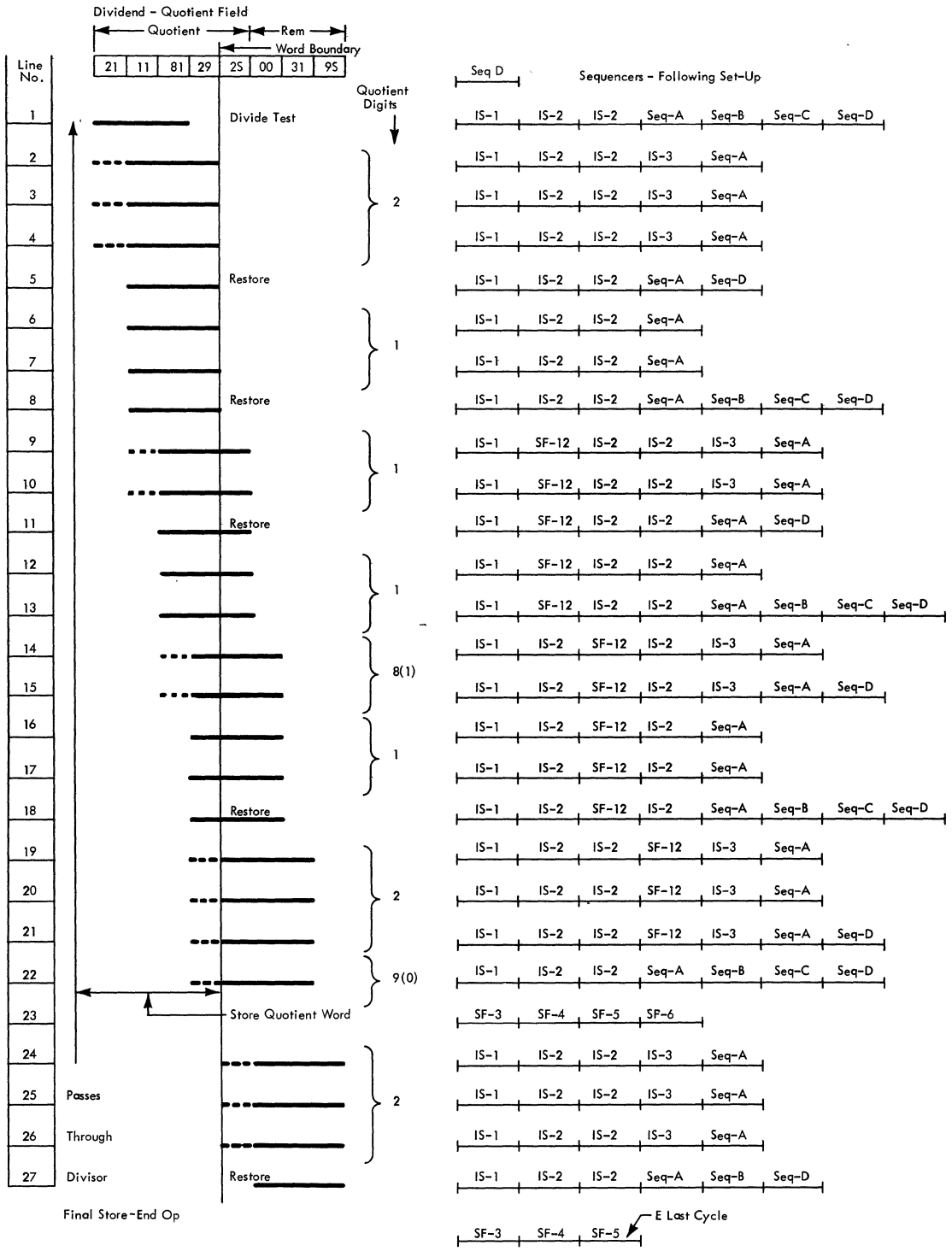
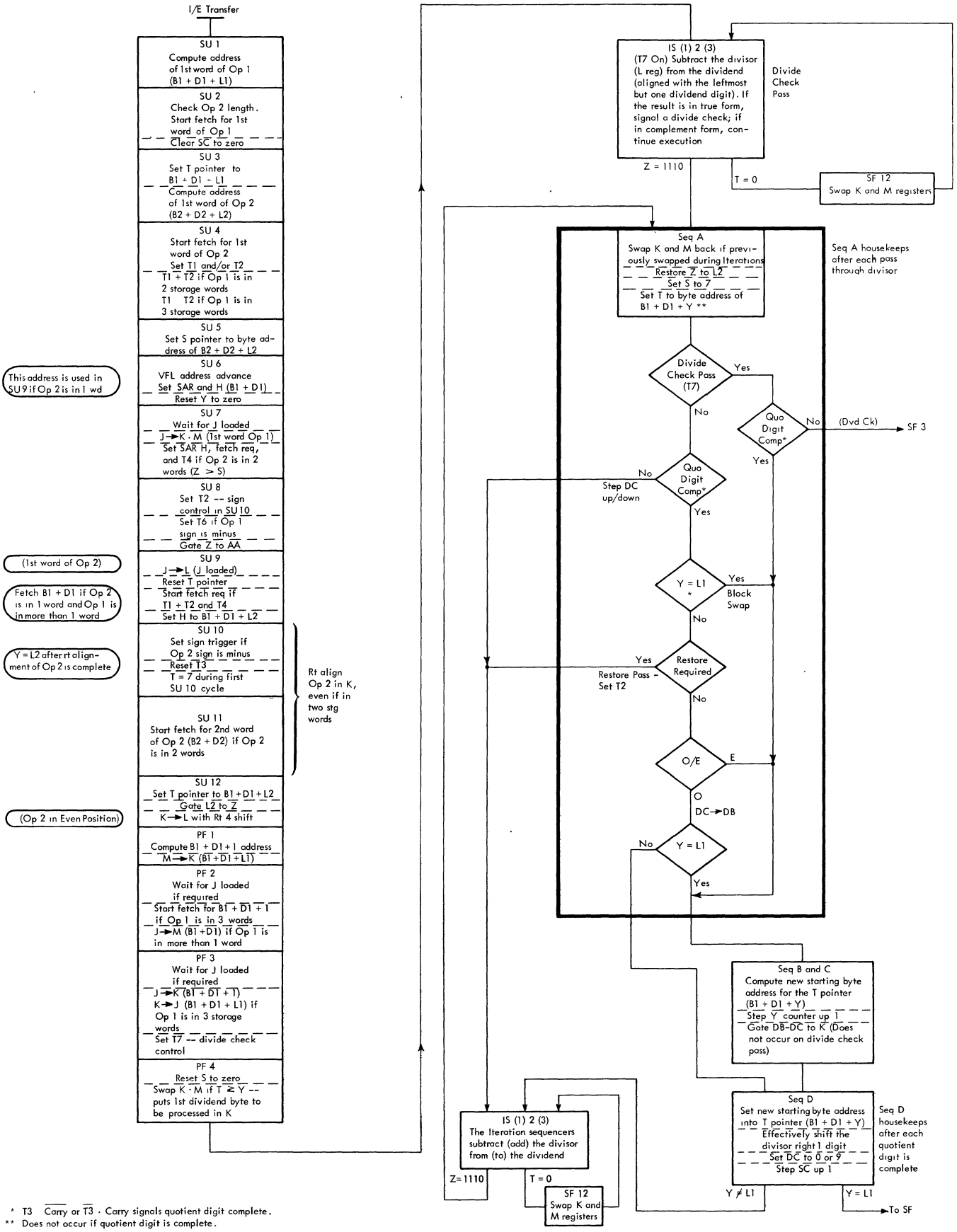


FIGURE 61. DIVIDE INTIRATIONS EXAMPLE



* T3 $\overline{\text{Carry}}$ or $\overline{\text{T3}}$ · Carry signals quotient digit complete.
** Does not occur if quotient digit is complete.

FIGURE 62. DECIMAL DIVIDE (DP) SIMPLIFIED EXECUTION SEQUENCE--2075

divisor. T5 is set to block the swap of K and M registers between even passes.

T5 is set during the IS 3 cycles in which T IN = 7. When T IN = 7 during IS 3, the last quotient byte of a dividend word is being developed. IS 3 cycles occur on odd passes through the divisor to accommodate a high-order carry-out from the decimal adder; therefore, T5 is not allowed to block K and M register swaps that occur during odd passes. During subsequent even passes through the divisor, the low-order digit of the quotient byte is generated. Each even pass through the divisor steps to the end of the dividend word (byte 0), but not across the word boundary. Therefore, in this case, the K and M swaps that normally occur during Seq A of even passes are undesirable and and blocked by T5.

Lines 19-22 of Figure 61 show the conditions in which the T5 trigger is used. Lines 19-21 represent odd passes through the divisor. At the start of each pass, the low-order dividend word is in the K register and the high-order word is in the M register.

When the end of the divisor is reached, during odd IS 2 cycles, SF 12 cycle is entered and K and M registers are swapped. IS 3 cycle then occurs and gates the carry-out, if any, from the decimal adder into byte 7 of the high-order dividend word, now in K register. Sequence A cycle then swaps K and M back; this places the high-order dividend word in M and the low-order word in K, ready for the next pass. Odd passes repeat (lines 20 and 21) until the dividend changes sign; the quotient digit is then complete. During the last odd Seq A cycle (line 21), K and M are swapped, and because the quotient digit is complete, Seq D cycle follows. During Seq D the quotient digit in the DC, generated during the odd passes, is set into the DB to become the high-order digit of the quotient byte.

T5 trigger is set during the first odd IS 3 cycle in which T IN = 7 (line 19) and remains on until the completed quotient word is stored. T5 blocks K and M swaps only during even passes through the divisor.

During even passes in which the last quotient digit of a quotient word is generated, the dividend word boundary is approached but not crossed. Therefore, the swap of K and M registers is undesirable and is blocked by T5 trigger, even though the T4 trigger is still on. This condition is shown on line 22 of Figure 61.

T6 -- Dividend Sign Trigger: The T6 trigger is used to generate the correct quotient and remainder signs.

The T6 trigger is set during SU 8 cycle of set-up sequence if the dividend sign is minus. When set, T6 trigger remains on throughout the execution of the decimal divide instruction, and is reset with ELC.

During Seq B cycle, after the last quotient digit is generated, the quotient sign is inserted in the low-order quotient byte. During this cycle, the status of the T6 trigger is compared with the status of the minus sign trigger (divisior sign) and the correct quotient sign inserted.

The status of the dividend sign trigger also determines the sign of the remainder. During the IS 1 cycle that processes the low-order bytes of the dividend, the remainder sign is generated and inserted. If the dividend sign is minus, the trigger is on and causes a minus sign to be generated for the remainder.

T7 -- Divide Test Trigger: The T7 trigger provides gating control during the divide test pass through the divisor.

T7 is set during PF 3 cycle of the set-up sequence; then during the IS 1 and IS 2 cycles when the divisor is subtracted from the dividend, T7 being on blocks setting the AV sum into K.

After all bytes of the divisor have been subtracted from the dividend, the dividend should change sign and indicate quotient digit complete (T3 on and no carry-out of AV). Seq A follows the divide test pass through the divisor. When T7 is on during Seq A and quotient digit is complete, the divide iterations continue. If, however, the dividend did not change sign, and quotient digit is not complete, T7 being on during Seq A causes a divide interrupt.

T7 trigger is reset during the following Seq C cycle.

T8 -- Terminate Trigger: As each quotient byte is completed and set into the K register, the mark register for that byte is set. After the last quotient digit is generated and the division completed, the remainder must also be stored; therefore, the mark register positions that correspond to remainder bytes must also be set. To set the marks for the remainder, a restore pass is forced, even if it is not needed. If a restore pass is forced when not needed, the results are not set into the K register.

When the remainder is in two words (T4 on) two restore passes are necessary. The first restore pass sets the marks for the high-order remainder word. The SF sequence then stores the high-order remainder word. The second restore pass then sets the marks for the low-order remainder word and the word is stored.

T8 trigger is set during the store of the high-order word if the T4 trigger is on. During the restore of the low-order word, T8 being on when the word boundary is reached, causes the word to be stored and the operation terminated.

Non-Restore Trigger: The non-restore trigger controls the restoring of the dividend during the decimal divide iteration sequence. Prediction of

the approximate value of a quotient digit enables a choice of the faster method, either restore or non-restore, to develop it.

Assume that the first quotient digit is generated by subtracting the divisor from the dividend until the remainder is negative. If the next quotient digit is in the range of 0 to 4, the dividend should be restored to positive, the divisor shifted right, and the next quotient digit generated by subtracting the divisor from the dividend. If, however, the next quotient digit is in the range of 5 to 9, the divisor should be shifted and the quotient digit developed by adding the divisor to the complement dividend until the dividend becomes positive.

When the dividend goes positive, the next quotient digit is predicted and the same decisions made. If the predicted quotient is in the range of 0 to 4, the dividend remains in true form, the divisor is shifted and the quotient digit generated by subtracting the divisor from the dividend. If the predicted quotient digit is 5 to 9, the dividend should be complemented, the divisor shifted and the quotient digit generated by adding the divisor to the dividend.

System/360 Model 75 predicts the next quotient digit by comparing the high-order divisor digit with the dividend digit that aligns with it. The divisor is added to or subtracted from the dividend one byte at a time and moves from right to left through the divisor.

For each divisor byte processed, the high-order nonzero digit is compared to the corresponding digit of the dividend. If the high-order divisor digit is zero, the low-order digit is compared to the corresponding dividend digit. If both digits of the divisor byte are zero, no comparison is made. The non-restore trigger is set or reset each cycle, depending on the value of the digits compared. When both digits of the divisor bytes are zero, the status of the non-restore trigger remains unchanged.

When the quotient digit is complete, the dividend has changed signs. The status of the non-restore trigger then determines whether the dividend should be restored. If the non-restore trigger is on, the dividend should not be restored; if it is off, the dividend should be restored.

Figure 63 shows conditions for the set and reset of the non-restore trigger.

Execution -- Decimal Divide

Execution of the decimal divide instruction involves three sequences; set-up, iteration, and store-fetch.

The set-up sequence fetches both operands, the dividend (Op 1) and the divisor (Op 2), from storage and aligns them in working registers. During the set-up sequence, the divisor and dividend signs are checked and controls are set to provide correct signs for the quotient and remainder.

Divide iteration follow the set-up sequence. During the divide iteration sequence, the quotient and remainder digits are generated.

The store-fetch sequence follows the iteration sequence to store the completed quotient-remainder in the Op 1 location of storage. The decimal divide instruction is always terminated by a SF sequence; however, store-fetch sequences are also interleaved with iterations if the quotient-remainder is in more than one storage word. When a quotient-remainder word is completed, a store-fetch sequence stores the word.

Set-Up Sequence

The VFL set-up sequence for decimal divide fetches the dividend and divisor words from storage to the K, M, and L registers. The dividend is fetched to the J, K, and M registers. If the dividend is in more than one storage word, the low-order dividend words are in the J and M registers and the high-order word is set into K, ready to start divide test. The divisor is fetched to the L register. If the divisor is in more than one storage word, both words are fetched and the divisor is right aligned in the L register. Because the divisor is allowed to contain a maximum of eight bytes, the L register contains the complete divisor at the end of set-up.

In addition to fetching both operands from storage, the set-up sequence makes a specification check, checks the signs of dividend and divisor, and sets controls used during the iteration sequence.

The set 12 sequence uses VFL sequencers SU 1. through SU 2 and PF 1 through PF 4 (Figure 6466).

Prior to the I/E transfer and during ELC of the previous instruction, L1 and L2 are set into Y and Z counters from IOP; L1 is set into Y and L2 into Z.

SU 1 Cycle: SU 1 cycle follows the I/E transfer. During SU 1 cycle, Y (L1) is gated to the AA preparatory to computing the storage address B1 + D1 + L1 of the low-order dividend (Op 1) word. Although the storage address is normally computed during the following SU 2 cycle, Y is gated to the AA during SU 1 cycle to overcome line delay.

If CPU is in single cycle mode during SU 1 cycle, the storage address B1 + D1 + L1 is computed and the VFL fetch request trigger is set to fetch the first dividend word from storage.

SU 2 Cycle: SU 2 cycle computes the storage address B1 + D1 + L1 of the low-order dividend word and initiates the fetch request to get the word from storage.

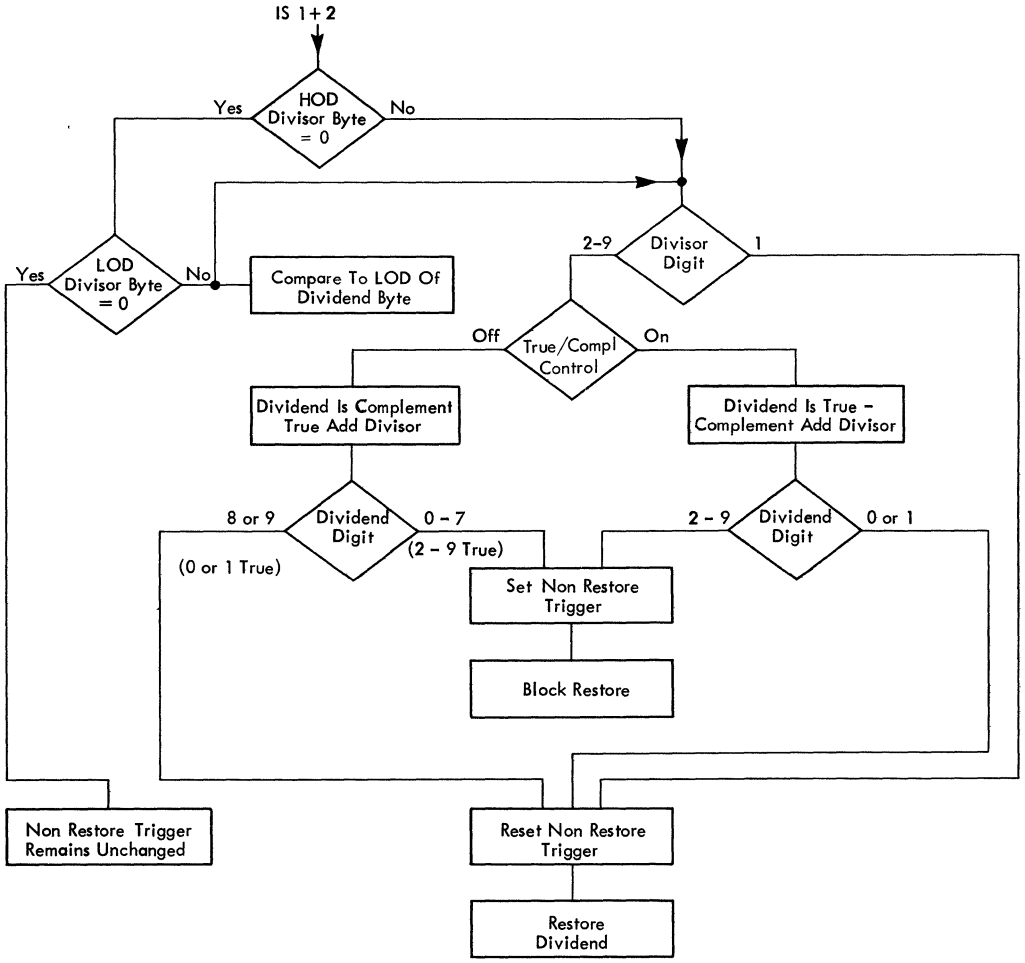


FIGURE 63. DECIMAL DIVIDE DECODE--NON-RESTORE

Dividend and divisor lengths are checked during SU 2. If the divisor length is greater than the allowable eight bytes, L2 $>$ 7, or if the divisor is equal to or greater than the dividend length, L2 $\geq$ L1, a specification interrupt occurs; the divide instruction is terminated by suppressing the remainder of the set-up sequence and going to the store-fetch sequence.

If CPU is in single-cycle mode during SU 2 cycle, the low-order dividend word has arrived from storage and is in the J register. The dividend word is transferred from the J register through the main adder to the K and M registers. Because the fetch request for the low-order dividend word is made during SU 1 cycle, single-cycle mode suppresses the fetch normally made during SU 2.

SU 3 Cycle: SU 3 is the accept wait cycle for the fetch request mode during SU 2. The ON status of the VFL fetch request trigger sets the BCU fetch request trigger at the beginning of SU 3 cycle. If the selected storage is not busy, an accept signal is returned to the E unit and enables the set-up sequence to continue. If the selected storage is busy, then the accept signal is delayed until storage priority is established; in which case, the set-up sequence remains in SU 3 cycle until accept arrives.

The storage address (B1 + D1 + L1) computed during SU 2 cycle is set into SAR and H registers at the beginning of SU 3 cycle. During SU 3, the byte address of the low-order digit of the dividend, contained in H register positions 21-23, is set into the T pointer. The T pointer is used later, during set-up, when the dividend sign is checked.

VFL address advance signal is sent to the I unit during SU 3 cycle preparatory to computing the storage address of the low-order divisor word. VFL address advance signals the I unit to gate B2 and D2 to IOP during the next cycle.

The Z counter provides the L2 factor to compute storage address B2 + D2 + L2 during SU 4 cycle. To overcome line delay and assure L2 is available

during SU 4, the gating of Z to the AA is started during SU 3 cycle.

SU 4 Cycle: SU 4 cycle (decimal divide) computes the storage address of the low-order divisor word (B2 + D2 + L2) and starts the VFL fetch request to get the divisor from storage. Because VFL address advance is gated to the I unit during SU 3 cycle, the I unit gates the B2 and D2 to IOP and thus, the AA during SU 4. Z is gated to the AA to provide L2. In the AA, B2 + D2 + L2 storage address is computed; the address is then gated to SAR and H registers.

At the same time the storage address is computed and the fetch request is made for the divisor word, the dividend length and the number of storage words that contain the dividend are checked. VFL triggers T1 and T2 are used during divide set-up to indicate the number of storage words that contain the dividend.

The dividend may be contained in 1, 2, or 3 storage words depending on its length and low-order byte address. For example, a dividend 4 bytes long (Y = 3) is contained as one storage word if the low-order byte address is 3 or greater ($T \geq Y$); whereas, the dividend is contained in two storage words if the low-order byte address is less than 3 ($T < Y$). A dividend greater than 9 bytes long ($Y > 8$) is contained in two or three storage words depending on the low-order byte address. Y counter and T pointer are compared to determine the dividend length and storage word relationship. Y positions 1, 2, and 4 are compared to T positions 1, 2, and 4. If T is less than Y, VFL T1 trigger is set to indicate that the dividend crosses a word boundary. Y greater than 8 indicates the dividend is 9 bytes or greater in length and therefore is contained in at least two storage words. VFL trigger T2 is set when Y is greater than 8. When VFL T1 or T2 trigger is on, the dividend is contained in two storage words. When VFL T1 and T2 are on, the dividend is contained in three storage words. VFL T1 and T2 control the fetch requests made later during set-up, for remaining dividend words.

SU 5 Cycle: SU 5 cycle (divide) is the accept wait cycle for divisor fetch request made during SU 4 cycle.

The B2 + D2 + L2 divisor storage address computed during SU 4 cycle is set into SAR and H registers at the beginning of SU 5. The low-order byte address of the divisor, H 21-23, is set into the S pointer. The S pointer is used later in SU 10, to control divisor alignment.

If storage priorities delay an accept to the fetch request made during SU 4, the set-up sequence remains in SU 5 until the accept arrives.

If CPU is in single-cycle mode, the divisor arrives from storage and is set into the J register before SU 5 starts. Therefore, during single-cycle mode, the divisor is transferred from the J register through the main adder to the L register.

SU 6 Cycle: SU 6 cycle sets VFL gating controls and sends VFL address advance to the I unit preparatory to computing the storage address and starting a fetch to get the second divisor word, if necessary. The gate L with S and the T decode out triggers are set to enable divisor and dividend sign control later.

Y and Z counters are reset. Y counter is reset because it counts from 0 up during SU 10. Z counter is reset because Y and Z share a common reset; Z is restored in SU 7.

SU 7 Cycle: When the low-order dividend word arrives from storage, it is set into the J register and the J loaded latch is set. During SU 7 cycle, the dividend word is transferred from the J register through the main adder to the K and M registers. SU 7 latch is enabled with J loaded; therefore, if the dividend word has not arrived, the set-up sequence remains in SU 7 until J is loaded.

In addition, during SU 7, L2 (IOP 12-15) is set into Z counter. S is compared with Z to determine if the divisor is in more than one storage word. If $S < Z$, the divisor is in two storage words, and the VFL fetch request trigger is set to get the second high-order divisor word from storage. When the divisor is in two words, VFL T4 trigger is set to remember that the divisor is in two words. Because the VFL address advance signal is sent to I unit during SU 6 cycle, storage address B2 + D2 + 0 is computed during SU 7 cycle and set into SAR and H registers.

VFL T3 trigger is set during SU 7 cycle to enable sign control of divisor during the first SU 10 cycle.

SU 8 Cycle: SU 8 cycle checks the low-order digit of the dividend for a sign. The low-order dividend word is contained in the K register. The K byte selected by the T pointer is gated through the LBG. The low-order digit from the LBG is the dividend sign and high-order digit of the byte is the low-order dividend digit. LBG is gated to the AOE for parity checking, and the low-order digit is sign decoded. If the LBG sign is minus, VFL T6 trigger is set. T6 controls the quotient and remainder signs generated during the divide iterations.

Z (L2) is gated to the AA by SU 8 latch preparatory to computing storage address B1 + D1 + L2 during SU 9 cycle. Because of line delay this gate is brought up early.

SU 9 Cycle: When the low-order divisor word arrives from storage, it is set into the J register and the J loaded latch is set. If the J loaded latch is not set when SU 9 cycle starts, the set-up sequence waits in SU 9 until J is loaded. The low-order divisor word is then transferred from the J register through the main adder to the L register. If CPU is in single-cycle mode, the low-order divisor word is transferred to the L register during SU 5 cycle; therefore, the transfer during SU 9 is blocked.

Z (L2) is gated to the AA and storage address B1 + D1 + L2 is computed. AA output is gated to SAR and H registers. However, the B1 + D1 address in SAR is protected by blocking the set of SAR. B1 + D1 + L2 storage address is used later during divide test and iteration passes to align divisor and dividend.

If the divisor is in one storage word (T4 off) and the dividend is in more than one word (T1 or T2 on) a fetch request is made during SU 9 to get the high-order dividend word (B1 + D1). If the divisor is in two storage words, this indicates that the fetch for the second divisor word is made during SU 7 and that the word has not arrived from storage by SU 9 time; therefore, the fetch for the high-order dividend word is delayed until SU 11 cycle (see Figure 6467).

The T pointer is reset to zero during SU 9 and gated to step down one. This sets the T pointer to 7 during the first SU 10 cycle.

SU 10 Cycle: The divisor sign is checked and the divisor right aligned during SU 10 cycles. SU 10 cycles repeat, aligning one divisor byte each cycle until all divisor bytes are processed.

The low-order divisor word is set into the L register at the beginning of the first SU 10 cycle. The L register byte selected by the S pointer is then gated through the RBG, through the decimal adder (AV) and into the K register byte selected by the T pointer. The S pointer is set to the low-order byte address of the divisor during SU 5. The T pointer is stepped down from 0 to 7 at the beginning of the first SU 10 cycle. Therefore, the low-order divisor byte is transferred from its location in the L register, indicated by the S pointer, into byte 7 of the K register. S and T pointers are stepped down each SU 10 cycle and the next higher order divisor byte transferred from the L register to the K register. SU 10 repeats until all divisor bytes are in the K register.

The divisor length code (L2) is set into the Z counter during SU 7. During each SU 10 cycle Z is stepped down. When Z steps to 1110, all divisor bytes have been aligned in the K register and the set-up sequence continues to SU 12.

During the first SU 10 cycle VFL T3 trigger is on. If T3 is on, the low-order digit from the RBG is checked for a sign. If the sign is minus, the minus sign trigger is set. The minus sign trigger is used during divide iterations to generate the correct quotient sign.

If the divisor is in two storage words, the S pointer steps to zero before the Z counter reaches 1110. In this case, the second divisor word must be set into the L register before SU 10 cycles can continue. When S = 0, SU 10 cycles are suspended and SU 11 entered. The fetch for the second divisor word is started during SU 7 cycle. The second divisor word returns from storage to the J register and sets the J loaded latch. If J is not loaded when entering, SU 11 cycles repeat until J is loaded. The divisor word is then transferred through the main adder to the L register. SU 10 is set again and the alignment continued until Z steps down to 1110.

At the same time Z counter steps down, Y counter steps up. This occurs because the Y counter controls dividend-divisor alignment during iterations and must contain the divisor length code (L2) when the iterations start, and because no direct path exists in CPU to gate L2 to Y. Therefore, Y is reset to zero during SU 6 and stepped up during SU 10. When Z = 0, Y = L2.

SU 11 Cycle: SU 11 cycle occurs only if the divisor is in two storage words. SU 11 transfers the second divisor word from J register to L register and restarts SU 10 cycles to complete divisor alignment.

When both divisor and dividend are each in more than one storage word, the fetch for the second dividend word (B1 + D1), normally made in SU 9, is delayed until SU 11.

During SU 11 the storage address B1 + D1 is computed and the VFL fetch request set. The output of the AA is gated to SAR and H; however, to retain the B1 + D1 + L2 address in H, the setting of H register is blocked.

SU 12 Cycle: SU 12 cycle follows SU 10 after all bytes of the divisor are right aligned in the K register. SU 12 cycle transfers the divisor from K to L register through the main adder. The main adder is gated to shift the divisor right 4 bit positions. The right 4 shift causes the divisor sign digit to be removed and the divisor right aligned when set into L register.

L2 is gated from IOP 12-15 into Z counter. H register positions 21-23 are set into the T pointer. This is the byte address of B1 + D1 + L2, and enables the T pointer to gate the correct dividend bytes from K register during divide test.

PF 1 Cycle: PF 1 cycle transfers the low-order dividend word from M to K register. One is gated to AA 28 preparatory to computing B1 + D1 + 1 address used during PF 2 to fetch the next dividend word, if necessary.

PF 2 Cycle: PF 2 cycle resets the SC to zero. The SC is used during divide iterations to define the odd-even passes through the divisor.

If T1 and T2 triggers are off, no other functions are performed during PF 2.

If either T1 or T2 trigger is on, the dividend is in more than one storage word. The fetch for the high-order dividend word is started during SU 9 or SU 11. When this dividend word arrives from storage to the J register, the J loaded latch is set. If J is not loaded, PF 2 repeats until J is loaded. The high-order dividend word is then transferred from J register through the main adder to M register.

If both T1 and T2 triggers are on, the dividend is in three storage words. A 1 is forced to AA 28 and storage address B1 + D1 + 1 is computed and gated to SAR and H. To retain B1+D1+L2 in H, the setting of H is blocked, VFL fetch request is set and the fetch started to get the last dividend word.

PF 3 Cycle: The functions of PF 3 cycle are controlled by VFL T1 and T2 triggers. If either or both T1 and T2 are off, PF 3 cycle performs only the function of resetting the S pointer and setting VFL T7 trigger. The S pointer is reset to zero and later stepped down to 7 so that the low-order divisor byte is gated from the L register during divide test. VFL T7 is set to define the first pass through the divisor as the divide test pass. The set-up sequence then advances to PF 4 cycle.

If T1 and T2 triggers are both on at the beginning of PF 3 cycle, the dividend is in three words. The fetch for the third dividend word is requested during the preceding PF 2 cycle. PF 3 waits until the J loaded latch is set, a signal that this dividend word has arrived in the J register. At the beginning of PF 3, the two other dividend words are contained in the K and M register; the low-order word (B1 + D1 + L1) is in K register, and the high-order word (B1 + D1 + 0) is in M register. Because divide iterations process the high-order dividend words first, and because the K and M registers contain those words currently in process during iterations, alignment of the dividend in K and M is started during PF 3.

When J is loaded, it contains dividend word B1 + D1 + 1. K and J registers are swapped by gating K through the main adder to J and J through RBL to K. Thus, the low-order dividend word (B1 + D1 + L1) is in J and the two high-order words (B1 + D1 and B1 + D1 + 1) are in M and K registers at the beginning of PF 4 cycle.

PF 4 Cycle: PF 4 is the last cycle of the divide set-up sequence. Final positioning of the dividend words occurs during PF 4 cycle. If the dividend is in one word, this word is in the K register when PF 4 cycle starts and therefore requires no relocation. If the dividend is in two or three storage words (either T1 or T2 on, or both T1 and T2 on), the first two words to be processed are in the M and K registers; the high-order word in M and the low-order word in K.

The divide test pass that follows set-up subtracts the divisor from an equivalent number of high-order dividend bytes. The byte address (B1 + D1 + L2) contained in the T pointer selects the dividend byte that aligns with the low-order divisor byte. A test is made during PF 4 cycle to determine which of the two registers, K or M, contains the dividend byte used to start the divide test. The divisor length (in Y counter) is compared with the byte address of the selected dividend byte (in T pointer). If $Y \leq T$, the selected dividend byte is in the high-order word contained in the M register. Because M is not a byte addressable register, the contents of M register is transferred to K register. At the same time, to retain the low-order dividend word, the contents of K register is transferred to M; M register is gated through the main adder to K, and K is gated through RBL to M. In this manner K and M are swapped.

When Y is greater than T, the selected dividend byte is in the lower order dividend word in K register. In this case, the dividend words are correctly positioned in K and M registers and no K and M swap occurs.

PF 4 is the last cycle of the decimal divide set-up sequence and set IS 1 sequencer to start the divide test pass.

Iteration Sequence -- Decimal Divide

Decimal divide iterations subtract the divisor from the dividend the number of times necessary to reduce the dividend to zero or to a value less than that of the divisor. The number of times the divisor is subtracted from the dividend is the quotient. When the dividend becomes less than the divisor, the divide instruction is terminated.

Starting at the high-order end of the dividend, the divisor is subtracted the number of times required to cause that portion of the dividend to go negative. The number of times the divisor is subtracted without causing the dividend to go negative represents the first quotient digit. The divisor is then shifted right one digit and the process repeated to generate the next quotient digit. As each quotient digit is generated, the divisor is shifted and the dividend reduced. This is repeated until the dividend becomes

zero or less than the divisor; the divide process is then terminated.

In System/360 Model 75 the divisor is subtracted from the dividend one byte at a time. The divisor is first aligned with the high-order end of the dividend, then, starting with the low-order divisor byte, each divisor byte is subtracted, one at a time, from the corresponding dividend byte until the high-order end of the divisor is reached. If the dividend remains positive, the divisor is again subtracted from the dividend, and again starting with the low-order byte of the divisor and stepping through the divisor and dividend until the high-order divisor byte is subtracted.

A quotient counter counts the number of times the divisor is subtracted from the dividend without causing the dividend to change sign. When the dividend changes sign, the quotient digit is complete and the value in the quotient counter is stored in the high-order digit position of the dividend field. The divisor is then shifted right one digit and the same process repeated to develop the next quotient digit. When the divisor has been shifted right until the low-order divisor byte aligns with the low-order dividend byte, the last quotient digit is being generated. When the low-order divisor and dividend bytes align and the dividend changes sign, the last quotient digit is complete. The quotient digit and the remainder are stored and replace the dividend. The decimal divide instruction is then terminated.

During decimal divide set-up sequence, the dividend and divisor are brought from storage into the J, K, M, and L registers. The dividend is in the K and M registers and the divisor is right-aligned in the L register. During the divide iterations, the divisor bytes are gated from the L register through the RBG to the TC + 6 input of the decimal adder (AV). The dividend bytes are gated from the K register through the LBG to the left side input of the decimal adder (AV). The divisor byte is subtracted from the dividend byte by complement adding (decimal 9's complement) in the AV. The output byte of the AV is then gated back to the K register. In this manner, each divisor byte is subtracted from the corresponding dividend byte during each pass through the divisor.

The digit counter (DC) is the quotient counter in which each quotient digit is generated. The DC is stepped each time a pass is made through the divisor.

Iteration sequencers IS 1, IS 2, and IS 3 are used during the execution of the decimal divide instruction to perform the decimal arithmetic, and VFL sequencers A, B, C, D, and SF 12 are used to set controls and align the divisor and dividend between passes through the divisor (Figure 6467).

IS 1 Cycle: IS 1 is the first sequencer in every pass through the divisor when subtracting (adding) from the dividend. IS 1 controls the gates to AD for sign control and hot 1 for subtraction.

IS 2 Cycle: IS 2 is the sequencer after IS 1 during which L bytes are subtracted (added) from K bytes. It loops on itself until coming to a word boundary or the end of the divisor field.

IS 3 Cycle: IS 3 is used to process an extra byte during an odd cycle pass through the divisor. The extra byte is necessary because the next high-order digit of the dividend may not be zero. The extra cycle is not necessary during even cycles because the high-order divisor digit is the low-order digit of a byte.

SF 12: SF 12 is used to swap K and M when a boundary is crossed during a pass through the divisor.

Seq A: Seq A is the first sequence following a subtraction (addition) of the divisor. It resets S, gates L2 to Z and gates H (B1 + D1 + Y) to T. If the portion of the dividend presently spanned by the divisor is in two words, K and M are normally swapped at this time. DC is stepped during Seq A if the quotient digit is not complete.

If the quotient digit is not complete, another subtraction (addition) must be made, IS 1 is set. If the quotient digit is complete, the next even cycle sequence is Seq B; next odd cycle sequence is Seq D.

Seq B, Seq C: When the even cycle digit is complete the new Y value must be added to B1 + D1. Seq B and Seq C gate Y to AA and release H.

Seq D: Seq D shifts L right 4 following odd cycles and left 4 following even cycles. The shift counter is incremented during Seq D. Seq D also gates H to T to get the new T starting point if Y was stepped and added to B1 + D1.

Figure 61 shows an example of divide iterations.

Store-Fetch Sequence -- Decimal Divide

The store-fetch sequence for the decimal divide instruction stores the completed quotient-remainder words as each is completed. Because all operand fetches are made during the divide set-up sequence, the store-fetch sequence only stores.

After the high-order quotient-remainder word is complete, and after each succeeding lower-order word is complete, the store-fetch sequence is entered and the completed word is stored at the Op 1 location in storage, with but one exception, the store-fetch sequence starts with SF 3 and sequences through SF 5 or SF 6 (Figure 6468). If the last quotient-remainder word is complete, or the E-

interrupt trigger is set, SF 5 terminates the instruction. If the decimal divide instruction is not complete, the SF sequence proceeds through SF 6 and back to IS 1 to continue.

SF 2 starts the instruction terminating sequence when a decimal divide specification violation is detected during SU 2 cycle of the set-up sequence. This occurs because the VFL sequence triggers and latches are used for both set-up sequences and for SF sequences. The VFL SF trigger defines which sequence is executed. When the VFL SF trigger is off, the VFL sequence triggers and latches perform set-up functions. When the VFL SF trigger is on, the VFL sequence triggers and latches perform store-fetch functions. Therefore, if a specification interrupt occurs during SU 2 cycle, the setting of VFL SF trigger is gated. The SF trigger is set at the beginning of the next cycle, SU 3. Because the SF trigger is set, VFL gating is changed from set-up to SF and SU 3 cycle is changed to SF 2. The SF sequence then progresses to SF 5 and terminates.

Storage Addressing

When the SF sequence is entered to store a completed quotient-remainder word, the storage address for that word is computed. (See Figure 61) VFL triggers T1 and T4 are used to determine which of the three possible storage addresses is computed.

VFL T4 trigger is the K and M swap trigger that is set when a dividend word boundary is crossed during iterations. If VFL T4 is on during SF sequences, it indicates that at least one quotient-remainder word remains to be processed and the storage address of the high-order word is computed.

VFL T1 trigger is used to remember that a SF sequence has occurred. It is set during the first SF sequence and, with T4, controls address generation for subsequent SFs.

VFL T1 and T4 triggers control gates to the AA during the SF 3 (cycle 1) when the storage address is computed. If T4 is off, Y counter is gated to AA to compute storage address B1 + D1 + Y. When T4 is on, the status of T1 determines the storage address computed. When T1 and T4 are on, a 1 is forced to input position 28 of the AA to compute storage address B1 + D1 + 1. Figure 60 shows the relationship of T1 and T4, and the quotient-remainder word to be stored.

Line 2 in Figure 60 indicates that the quotient remainder is in one storage word. T4 trigger is off; therefore, the quotient-remainder is stored at storage address B1 + D1 + Y. In this case, the first SF sequence also terminates the instruction.

Lines 3, 4, and 5 indicate a quotient-remainder in two storage words. Two passes through the SF sequence are required, one to store each quotient-remainder word. The first SF sequence stores the high-order quotient-remainder word at storage address B1 + D1, then iteration sequencing is started again to process the low-order quotient-remainder word. When the low-order word is complete, the second SF sequence is started and the low-order word stored at address B1 + D1 + Y. T4 trigger is on during the first SF sequence because a word boundary is crossed during the iterations when the first quotient-remainder word is processed. During the iterations that follow the store of the high-order word, if the divisor is less than 8 bytes long ($Z < 7$), the portion of the dividend in process is contained within one word. No word boundaries are crossed; therefore, T4 is reset. T1 trigger is on when the second SF starts; however, T4 being off overrides the status of T1 and gates Y to the AA. The second quotient-remainder word is stored at address B1 + D1 + Y and the instruction is terminated.

Lines 6, 7, and 8 of Figure 60 indicate conditions wherein the dividend and, therefore, the quotient-remainder is contained in three storage words. Because a word boundary is crossed when the high-order quotient-remainder word is developed, T4 trigger is on when the first SF sequence starts. T4 trigger on and T1 off causes storage address B1 + D1 to be computed during SF 3 cycle. The first SF sequence sets T1 trigger, which remains on until the instruction is terminated. The second SF sequence stores word B1 + D1 + 1. When T1 and T4 triggers are on, a 1 is forced to input position 28 of the AA and storage address B1 + D1 + 1 is computed. T4 trigger is off when the last SF sequence starts; therefore, the storage address (B1 + D1 + Y) of the low-order quotient-remainder word is computed.

Store-Fetch Cycles

When the last quotient digit of the quotient-remainder word is complete, a SF sequence is started to store the completed word in the Op 1 location in storage. See Figure 6468.

SF 3 Cycle: SF 3 is the first cycle of the SF sequence. During SF 3 cycle, the storage address is computed and the VFL store request trigger is set.

T1 trigger is set to remember that the first SF sequence has occurred. The status of T1 and T4 triggers controls address generation during subsequent SF sequences.

A set to VFL T8 trigger is gated during this cycle if the last quotient digit has been developed (Y = L1) and a word boundary separates the quotient and remainder. The set status of T8 then forces a dummy restore pass to set the marks for the

remainder bytes in the next lower-order quotient-remainder word.

If the E interrupt trigger is on when entering SF 3, the setting of the VFL store request trigger is blocked and the VFL end sequence trigger is set to terminate the instruction.

SF 4 Cycle: SF 4 is the accept wait cycle of the SF sequence. The VFL store request trigger, set during SF 3, causes the BCU store request trigger to be set at the beginning of SF 4. If either the selected storage is not busy or a higher priority does not exist, an accept signal is received at the E unit and the SF sequence proceeds to SF 5. Otherwise, the SF sequence waits in SF 4 until the accept arrives.

If the E interrupt trigger is on, a store request is not made. In this case, SF 4 cycle does not wait for accept.

VFL end sequence trigger on during SF 4 gates the setting of ELC trigger.

SF 5 Cycle: SF 5 terminates the instruction if ELC trigger is set at the beginning of SF 5. Otherwise, SF 5 starts preparations to return to iteration sequence and continue processing.

Y counter is gated to AA to compute storage address B + D + Y; this address is used later to set the T pointer during iterations.

Gates are established to transfer the low-order dividend word from M to K. The word in K at the beginning of SF 5 is the quotient-remainder word being stored during this SF sequence; therefore, the next lower-order dividend word is the next to be processed.

If the divisor contains less than 8 bytes, no word boundaries will be crossed during subsequent iteration cycles prior to the next SF sequence; therefore, VFL T4 trigger is reset during SF 5.

The block-swap (T5) trigger is reset during SF 5 to enable normal K and M swaps during following iteration cycles.

SF 6 Cycle: SF 6 is the last cycle of the SF sequence before return to iterations. Relocation of remaining dividend words before iterations start again are gated during SF 6 cycle. The objective is to place, in the K register, the word that contains the next dividend byte to be processed. A maximum of two words can remain to be processed.

When the dividend is in two storage words, one word remains to be processed. This word is gated from the M register into the K register during the preceding SF 5 cycle and set into K at the beginning of SF 6 cycle. Therefore, the last dividend word is correctly located and additional word transfers during SF 6 are unnecessary. J is gated to M but has no significance when the dividend is in two words.

When the dividend is in three storage words, and this is the first SF sequence, two dividend words remain to be processed. The word that contains the next storage byte to be processed is determined by the divisor length. When the divisor is less than 8 bytes (L2 < 7), the higher-order dividend word contains the next byte to be processed. In this case, the higher-order word is gated from M to K and the low-order word in J is gated to M. The two remaining dividend words are then properly positioned when iterations start again. If the divisor length is 8 bytes (L2 = 7), the low-order dividend word contains the next byte to be processed. In this case, the low-order dividend word is gated from J register to K register and the high-order word remains in M register.

When the dividend is in three storage words, gates during the second SF sequence are the same as if the dividend were in two storage words.

Iterations start again at the end of SF 6 cycle.

DECIMAL MULTIPLY

- Multiplication done similar to longhand method.
- Multiplier and multiplicand fetched from storage and aligned in L, M, and K registers during set-up sequence.
- Iteration sequence develops product by over-and-over addition of multiplier.
- Store-fetch sequence stores product words as they are completed.

Method of Multiplication

Decimal multiply is done in the System/360 Model 75 basically the same as the normal longhand method. An example of this longhand process is:

4976	multiplier
x23	multiplicand
14928	partial product
9952	partial product
114448	product

Note that the use of the multiplier and multiplicand is opposite to the way they are normally thought of. This provides consistency in terminology with that of other models of the System/360.

The entire multiplier is multiplied by each digit of the multiplicand and the results added to each partial product right aligned with the corresponding multiplicand digit.

The computer does each multiplication by adding the multiplier to itself a number of times equal to the multiplicand digit.

When the multiplicand digit is 6 or greater, the operation can be speeded up by multiplying the multiplier by 10 and subtracting the multiplier a number of times equal to 10 minus the multiplicand digit. The method is shown below.

Let A be the multiplier and B be the multiplicand digit. As a numerical example, consider A = 246 and B = 7:

AxB	= Ax10-Ax(10-B)	longhand check
	= 246x10-2x(10-7)	
	= 2460-246x(3)	246
	= 2460-738	x7
	= 1722	1722

Multiplying the multiplier by 10 is accomplished by adding 1 to the next high-order multiplicand digit.

The increase in speed by using this method is shown by the fact that it was necessary to subtract the multiplier three times, whereas, it would have taken seven additions.

In general, the multiply process in System/360 Model 75 is as follows:

The low-order multiplicand digit is set into a counter and is decoded. If it is 0 (10 following a subtraction), the multiplier is shifted left 1 digit and the next multiplicand digit is set into the counter. If it is not 0 (10), it is decoded for $D \leq 5$ or $D > 5$. This determines whether the multiplier will be added to or subtracted from the partial product.

The multiplier is then added (subtracted) to the partial product field a byte at a time. At the end of each addition of the multiplier, the counter is decremented (incremented) by 1 and decoded. The addition of the multiplier continues until the counter goes to 0 (10). Then the next multiplicand digit is set into the counter and the multiplier is shifted left 1 digit, etc. This continues until all multiplicand digits are exhausted.

Unit Functions

Registers

J Register: The J register contains the multiplicand field. J register bits 60-63 are set into the DC where the additions are counted. As each multiplicand digit is set into the DC, the content of J register is shifted right four bits to position the next multiplicand digit to be set into the DC after the present one is completed.

During the set-up sequence, the low-order multiplicand word is fetched from storage and right-aligned in the J register. The sign digit of the multiplicand is used to set a sign control trigger and then shifted out of the J register. During Seq D cycle, between set-up and the first iteration sequence, the first (low-order) multiplicand digit is set into the DC, then that digit is shifted out of the J register.

K and M Registers: The K register contains the portion of the partial product presently being accumulated. The M register contains the portion of the partial product on the other side of the word boundary, if a word boundary is crossed by the partial product.

The multiplicand-product field in storage may be as long as 16 bytes, therefore, up to two word boundaries may be crossed by this field. The portion of the product field worked on at any one time is limited by the length of the multiplier (a maximum of eight bytes). Therefore, two registers are sufficient to hold the portion of the partial product being worked on.

When a partial product word boundary is crossed while making an addition, the contents of the K and M registers are swapped. When the portion of the product in one storage word is completed, that word is stored.

L Register: The L register holds the entire multiplier field during execution of the decimal multiply instruction. The multiplier is right-aligned in the L register during set-up; thereafter, the multiplier is shifted right or left one digit as necessary during execution cycles.

VFL Counter and Pointer Functions -- Decimal Multiply

Y Counter: The Y counter is initially set with the length of the multiplicand-product field (L1) from IOP 8-11. When a byte of the multiplicand has been processed, the Y counter is stepped down by 1. Y > L2 indicates the multiplicand field contains significant data bytes still to be processed. When Y = L2, the product is complete; the remaining high-order bytes of the multiplicand are then checked for nonzero digits. When the Y counter steps to 0, all bytes of the multiplicand-product field have been processed and the multiply instruction is ready for termination.

Z Counter: The Z counter counts the multiplier bytes as they are processed. The Z counter is set with the length of the multiplier field (L2) at the beginning of each pass through the multiplier, then stepped down one as each multiplier byte is processed. When Z = 0, all bytes of the multiplier have been added to the partial product.

S Pointer: The S pointer is used to select the multiplier byte that is added to the partial product from the L register. The S pointer is reset to 0, then stepped down 1 to 7 at the beginning of each pass through the multiplier; then, as each multiplier byte is processed, the S pointer steps down 1 before selecting the next multiplier byte.

T Pointer: The T pointer selects the partial-product byte of the K register to be added to the multiplier byte to form a new partial-product byte. The T pointer also selects the K register byte in which the new partial-product byte is placed.

The initial setting of the T pointer is determined by the storage address computed and used to fetch the first multiplicand word (B1 + D1 + L1); this is the low-order partial-product byte address of the K register. The T pointer then steps down one during each cycle in which multiplier and partial-product bytes are added. Several passes through the multiplier may be required to process each multiplicand byte; therefore, after each pass, the T pointer is restored to the previous starting point unless a new multiplicand byte is to be processed. When a new multiplicand byte is selected for processing, a new starting point is established for the T pointer. In this manner, the correct byte alignment of multiplier to partial product is maintained throughout the execution of the instruction.

The starting byte address used for the T pointer is retained in H register positions 21-23. When the T pointer is restored to the starting point, H register positions 21-23 are gated to it. The Y counter is initially set with L1 and stepped down by 1 as each multiplicand byte is processed: each new starting point for the T pointer is determined by B1 + D1 + Y.

When the T pointer is set to 0 from B1 + D1 + Y, the last byte of a multiplicand word is being processed; when this multiplicand byte is completed, a new multiplicand word must be fetched and the completed product word stored.

Digit Counter: The DC is used to control the number of times the multiplier is added to or subtracted from the partial product during processing each multiplicand digit. The decoded outputs of the DC determine the add or subtract status and the completion of each multiplicand digit.

The multiplicand digit to be processed is set into the DC. Outputs of the DC are then decoded to determine if the digit is a value greater than 5. If $D \leq 5$, the multiplier is added to the partial-product and the DC is decremented by 1 at the end of each pass; if $D > 5$, the multiplier is subtracted from the partial-product and the DC is incremented by 1 at the end of each pass. When the DC steps to 0 or 10, the processing of the multiplicand digit is complete.

Odd and Even Cycle Definition: The basic addressable unit of data in the IBM System/360 Model 75 is the byte; in decimal multiply the basic unit of data is the digit. Because the multiplicand is processed one digit at a time, the effective shift of the multiplier relative to the partial product must be one digit at a time.

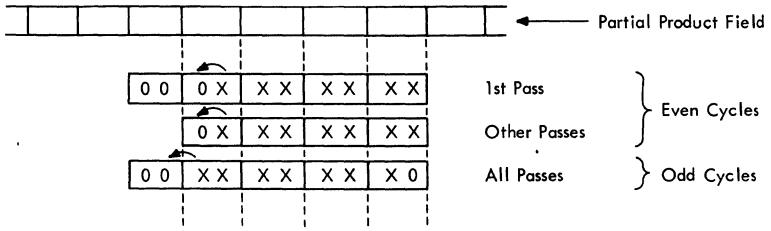
To shift the multiplier one digit to the left, the starting point of the T pointer is shifted left (reduced) by one byte and the multiplier in the L register is shifted right one digit. To process the next multiplicand digit, the multiplier in the L register is shifted left one digit without changing the starting point of the T pointer.

The shift counter is used to gate the odd or even cycle functions during execution of the decimal multiply instruction. The shift counter is set to 0 during the set-up sequence, and then incremented by 1 after each multiplicand digit is processed. Odd cycles are cycles in which the multiplier is shifted left 4; the value in the shift counter is odd. Even cycles are those in which the multiplier is shifted right 4; the value in the shift counter is even.

Control Trigger Functions

T2 Dummy Cycle Trigger: Multiplicand digits have numeric values of 0 through 9. When a 9 digit is processed following a subtract pass, the DC is incremented by 1 to 10. If the 10 digit is processed during an even cycle it is necessary only to shift the multiplier left one digit and proceed to the next multiplicand digit; if the 10 digit is processed during and odd cycle, IS 3 cycle must be entered to gate 99 into the K register byte to the left of the highest-order partial-product byte developed so far. To enter IS 3 cycle and at the same time step the T pointer to the correct byte location, a dummy pass is made through the multiplier without adding; VFL T2 trigger is set to block the addition during this dummy pass.

T3 True/Complement Trigger: The T3 trigger is used during execution of the decimal multiply instruction to control the true/complement inputs



↶ Indicates possible carries out of high-order position.

FIGURE 64. EXTRA IS 3 CYCLE - DECIMAL MULTIPLY

Example of a multiply odd addition cycle following a subtraction.					Example of a multiply odd subtraction cycle following a subtraction.				
Odd			9S	L - multiplier				5S	L - multiplier
		01	81	J - multiplicand			08	91	J - multiplicand
Odd							00	00	K
					Odd		00	5S	L
		00	00	K			00	5S	K
		00	9S	L					
		00	9S	K	Shift L		05		
Shift L		09					00	5S	K
					Even	99	95		L - complement
		00	9S	K		99	95	5S	K
	99	91		L - complement					
Even	99	91	9S	K	Shift L		50		
		91		L - complement					
	99	82	9S	K		99	95	5S	K
					Odd *	99	50		L - complement
Shift L		90				99	45	5S	K
	99	82	9S	K	Shift L	05			
	00	90		L					
Odd	00	72	9S	K		99	45	5S	K
	00	90		L	Even	05			L
	01	62	9S	K		04	45	5S	K

___ Extra Byte

* If the multiplicand was two bytes (3 digits), the last digit (8) would have been forced to be processed by addition. This leaves the product in true form.

FIGURE 65. EXTRA BYTE PROCESSING - DECIMAL MULTIPLY

to the decimal adder. If the multiplicand digit is 5 or less, the multiplier is added to the partial-product; T3 trigger is reset and the decimal adder performs true additions. If the multiplicand digit is 6 or greater, the multiplier is subtracted from the partial-product; T3 trigger is set and the decimal adder performs complement additions (subtracts).

T4 Swap Trigger: The T4 trigger is set to record that the contents of the K and M registers have been swapped. If a partial-product word boundary is encountered during a pass through the multiplier, the contents of the K and M registers are swapped (SF 12) and VFL T4 trigger is set to recall it. When the pass through the multiplier is complete, the set status of T4 trigger causes the contents of the K and M registers to be swapped back to the original state before the next pass through the multiplier begins. T4 trigger is then reset.

T5 Extra Cycle Trigger: The VFL T5 trigger is set to cause an extra IS 3 cycle at the end of the first pass of every even multiply pass in which the partial product is nonzero. This extra IS 3 cycle (Figures 64 and 65) sets 00 or 99 into the next higher-order partial-product byte.

After a multiplicand digit is processed by subtracting the multiplier from the partial-product, the partial product is in complement form; the high-order zeros should be 9's unless the partial product is zero. IS 3 cycles that occur at the end of every odd pass through the multiplier gate high-order carries into the next byte and complement the high-order zeros to 9's on subtract passes. Because the multiplier is shifted right during even passes, the last IS 2 cycle of each even pass propagates any high-order carries into the next digit position of the high-order partial-product byte, or complements this digit to 9 on subtract passes. This single high-order complement 9 is insufficient, however, to accommodate any high-order carries that may occur during subsequent odd passes. Therefore, VFL T5 trigger provides an extra IS 3 cycle to terminate the first pass of every even multiplicand digit if the partial-product is nonzero.

T6 Multiplicand Sign Trigger: The VFL T6 trigger is set during set-up if the multiplicand sign is minus. During the first IS 1 cycle, the status of T6 and of the minus sign trigger (multiplier sign) determines the sign of the product.

T7 First Digit Trigger: The VFL T7 trigger is used to provide product sign control gates during the first IS 1 cycle. T7 trigger is set during the set-up sequence, then reset after the first multiplicand digit is processed.

T8 Termination Trigger: The VFL T8 trigger is used as the termination trigger for decimal multiply; it provides gates to zero check the high-order multiplicand digits.

To insure that the multiplicand field in storage is large enough to receive the product, the multiplicand must contain a number of high-order zeros equal to the number of multiplier digits. Therefore, after all significant multiplicand digits are processed (Y = L2) the remaining high-order multiplicand digits must be zero checked.

When Y = L2, the T8 trigger is set. T8 trigger On causes sequence D cycles to repeat until Y Lth = 1110. During each sequence D cycle, a multiplicand digit is set into the DC and zero checked. If a nonzero digit is detected, a data interrupt occurs.

Set-Up Sequence (Figures 67 and 6474)

- Fetch first multiplicand word from storage and align in J register.
- Fetch all multiplier bytes from storage and align in the L register.
- Check length of both operands for specification error.
- Clear K and M registers to all zeros.
- Set VFL control triggers.
- Start Iteration Sequence.

The set-up sequence for Decimal Multiply fetches the multiplier and the low-order multiplicand word from storage. The multiplier may start at any byte location within a word and may cross a word boundary. During the set-up sequence, the entire multiplier is fetched from storage and right-aligned in the L register. All multiplier digits are validity checked as they are aligned.

The multiplicand digits from the low-order digit position of the J register are used during the multiply iterations. Therefore, the storage word containing the low-order multiplicand digit is fetched and right-aligned in the J register during the set-up sequence. Subsequent multiplicand words are correctly aligned when they arrive from storage into the J register.

The K and M registers are cleared to all zeros during the set-up sequence because the product is accumulated in these two registers during the iteration sequences.

During the set-up sequence the lengths of both operands are checked; if incorrect, the instruction is terminated with a specification interrupt.

The VFL sequencers used to execute the multiply set-up sequence are: SU 1 through SU 12, and PF 1 through PF 4.

SU 1 Cycle: Operand lengths L1 and L2 are set into the Y and Z counters from IOP register positions 8 - 15; this function is gated by ELC of the previous instruction.

The L1 factor is in the Y counter and is used to compute the multiplicand (Op 1) storage address. Then, the L1 factor is gated to the AA by SU L1. The computed storage address is not used until the following cycle. However, to overcome line delay, Y is gated to the AA early.

If in single cycle mode, the VFL fetch request is set to get the first multiplicand word from storage.

SU 2 Cycle: During SU 2 cycle the fetch request for the first multiplicand (Op 1) word is made. Setting the VFL fetch request trigger causes the B1 + D1 + L1 computed storage address to be set into the SAR and H registers.

The AEOB is gated to the SC and the SC is released; this sets the SC to zero. The SC is stepped and used later to control the odd and even passes during the iteration sequence.

In addition, SU 2 cycle checks operand length specifications. If a specification error exists, the instruction is terminated with a specification interrupt.

Z counter (L2) is compared to 7 and to Y counter (L1). If L2 > 7 or L2 ≥ L1, a specification error exists; setting the fetch request is blocked and the SF trigger is set. When the SF trigger sets, the function of the VFL sequence triggers is changed from set-up gating to store-fetch gating. Thus, the next machine cycle that would have been SU 3 becomes SF 2. SF 2 then starts the store-fetch sequence to terminate the instruction and signal the specification interrupt.

If no error condition exists, SU 3 cycle follows SU 2.

In single cycle mode, the fetch request for the first multiplicand word is made during SU 1 cycle. SU 2 cycle is then used to transfer the first multiplicand word from the J register to the K and M registers through the AM.

SU 3 Cycle: During SU 3 cycle the starting byte address of the multiplicand is set into the T pointer from H register positions 21-23.

Also, during SU 3 cycle, gates are established to the I unit before computing the storage address of the first multiplier (Op 2) word. VFL address advance is gated to transfer the third halfword of the instruction (B2 and D2) from the A/B registers into the IOP register for the following cycle. To overcome line delay and insure that L2 arrives at the input to the AA early enough in the next cycle, the Z counter is gated to the AA during SU 3.

SU 3 is the accept-wait cycle for the fetch request made during SU 2 cycle. If the selected storage is not busy, the BCU provides an immediate accept signal and SU 3 is only one cycle. If the selected storage unit is busy, the BCU delays the accept signal until storage becomes available; during this time, SU 3 cycles repeat until the accept signal arrives.

SU 4 Cycle: During SU 4 cycle, the VFL fetch request trigger is set to fetch the first multiplier word (Op 2) from storage. Setting the fetch request trigger gates the computed storage address of the first multiplier word, B2 + D2 + L2, from the AA to the SAR and H registers; SAR and H registers are set at the beginning of the next cycle.

Checks are made during SU 4 cycle to determine the number of bytes the first multiplicand word must be shifted to right align in the J register. Because multiplicand alignment occurs near the end of the set-up sequence, VFL control triggers T1 and T2 are used to record the number of bytes the multiplicand must be shifted. The starting byte address of the multiplicand, in the T pointer, is examined during SU 4 cycle. If T is less than 4, VFL T1 trigger is set to gate a right 32 shift (4 bytes) during SU 12 cycle. If the T pointer equals 3 or 7, VFL T2 trigger is set to indicate that the multiplicand is correctly aligned when the first PF 1 cycle starts.

SU 5 Cycle: This is the accept-wait cycle for the fetch request made during the previous SU 4 cycle. If storage priorities prevent an immediate accept, then SU 5 cycles repeat until an accept signal arrives from the BCU.

After the accept signal arrives, the multiplier starting byte address is gated from the H register positions 21 - 23 to the S pointer and the S pointer is released. Y counter (L1) is gated to the AA before re-computing the starting byte address of the multiplicand.

If, in single cycle mode, the first multiplier word has arrived from storage and is in the J

register; this word is gated from the J register through the main adder (AM) to the L register and set into the L register at the beginning of the next CPU clock cycle.

SU 6 Cycle: During SU 6 cycle the gating of Y counter to the AA is continued and the starting byte address of the multiplicand, B1 + D1 + L1, is computed in the AA. The output of the AA is gated to and set into SAR and H registers. This provides the correct byte address for the T pointer when the multiplicand is right-aligned during PF 1 cycles, and when iteration cycles start.

VFL address advance signal is gated to the I unit during SU 6 cycle before computing the storage address of the second multiplier word if it is needed.

Control triggers Gate L with S and T Decode Out are gated to set during SU 6 cycle. These two triggers are set at the beginning of the next cycle; they provide data gates to check the signs of the multiplier and multiplicand.

SU 7 Cycle: SU 7 is the cycle that transfers the first multiplicand word from the J register to K and M registers, and starts the fetch request for the second multiplier word, if the multiplier is in more than one storage word.

The functions of the SU 7 cycle, however, depend upon the set status of the J loaded trigger. If the first multiplicand word has not arrived from storage and the J loaded trigger is not set by SU 7 time, SU 7 cycles repeat until the J loaded trigger is set. The amount of delay involved, if any, depends upon the type of storage unit selected when the fetch request is made. If a high-speed storage is selected, no delay occurs because the multiplicand word arrives from storage and the J loaded trigger is set at the beginning of SU 6 cycle. If an LCS is selected, the set-up sequence remains in SU 7 for several cycles waiting for the J loaded trigger to set.

SU 7 gates the multiplicand word from the J register through the main adder (AM). When SU L7 is enabled by J loaded, the main adder out bus (AMOB) is gated to K and M registers and they are set at the beginning of the next cycle. If the machine is in single-cycle mode, the multiplicand word is transferred to K and M registers in SU 2 cycle instead of SU 7.

If the multiplier is contained in two storage words, the fetch request for the second word is made during SU 7 cycle. To determine that the multiplier is in two storage words, the starting byte address in the S pointer is compared to the multiplier length in the Z counter. If S is less than Z, the multiplier is in two storage words; the VFL fetch request trigger is set to fetch the second word of the multiplier, and control trigger T4 is set to record that the multiplier is in two words.

The gate to set the VFL fetch request trigger also gates the output of the AA to the SAR and H registers. Because VFL address advance was gated to the I unit during SU 6 cycle, storage address B2 + D2 is computed and gated to SAR and H during this cycle. The set of the H register is blocked, however, to retain the starting byte address of the multiplicand, B1 + D1 + L1.

SU 7 cycle also gates the set of VFL control trigger T3 to provide multiplier sign control during SU 10 cycle.

SU 8 Cycle: The multiplicand sign is checked during SU 8 cycle. Because the T decode out trigger is set at the beginning of SU 7 cycle, the T pointer selects the low-order multiplicand byte in the K register and gates it through the LBG. The gating of LBG to the AOE is active during SU 8 cycle to enable sign decoding of the low-order LBG digit, the multiplicand sign digit.

If the multiplicand sign is minus, VFL control trigger T6 is set. The status of T6 is used later to compare with the multiplier sign and determine the correct product sign.

SU 9 Cycle: The SU 9 cycle transfers the first multiplier word from the J register to the L register, resets the T pointer to zero, and gates the Z counter to step down 1.

The functions of SU 9 cycle are enabled by the set status of the J loaded trigger, as in SU 7 cycle. When the J loaded trigger is on during SU 9 cycle, the first word of the multiplier is gated from the J register through the main adder to the L register. If in single cycle mode, this occurs in SU 5 cycle.

If an invalid address or SAP error has occurred during previous fetches, the set-up sequence is terminated at this point. When SU L9 is enabled, the set status of either the invalid address or SAP error trigger gates the set of the SF trigger, the E interrupt trigger, and VFL sequence trigger 4; this combination causes a terminating store-fetch sequence to start at SF 3 cycle.

SU 10 Cycle: SU 10 cycles right-align the multiplier in the K register. One multiplier byte is transferred from the L register through the decimal adder to the K register each SU 10 cycle. SU 10 cycles repeat until all multiplier bytes are aligned in the K register.

The S and T pointers are used to select and right-align the multiplier bytes while the Z counter counts

the number of bytes processed, and signals the end of alignment. When SU 10 cycles start, the S pointer contains the byte address of the low-order multiplier byte in the L register; the T pointer contains a value of 7; it is reset to zero during SU 9, then stepped down 1 to 7 at the beginning of the first SU 10 cycle. The Z counter is set to the multiplier length, L2, at the beginning of the set-up sequence and steps down one as each multiplier byte is aligned. The T pointer and the Z counter are each stepped down 1 at the beginning of every SU 10 cycle. Except for the first SU 10 cycle, the S pointer is stepped down at the same time.

During each SU 10 cycle, the multiplier byte selected by the S pointer is gated from the L register through the RBG into the AV. The AV output is then gated into the K register byte selected by the T pointer. Detection circuits at the output of the RBG provide sign decoding and validity checking of each multiplier digit as each multiplier byte is aligned. Byte parity is checked in the AV. If an invalid decimal digit or incorrect byte parity is detected, the data check latch is set. This causes the E interrupt trigger to set and the instruction to terminate after PF 4 cycle.

The first SU 10 cycle gates the low-order multiplier byte into byte position 7 of the K register; this is the byte that contains the multiplier sign. VFL control trigger T3 is on during the first SU 10 cycle to enable sign decoding and detection of the low-order multiplier digit. If the multiplier sign is minus, the VFL minus sign trigger is set; T3 trigger is then reset at the end of the cycle.

The end of multiplier alignment is signalled when the Z counter steps down to 1110. Because the maximum allowable multiplier length is 8 bytes, all data bytes of the multiplier are right-aligned in the K register when the Z counter equals 1110, and the set-up sequence proceeds to SU 12 cycle.

If the multiplier is contained in two storage words, the S pointer steps down to zero before the Z counter equals 1110. During the cycle that S steps to zero, the last multiplier byte in the L register is transferred to the K register. The storage word that contains the remaining multiplier bytes must then be set into the L register to complete the alignment. When the S pointer equals zero before the Z counter steps to 1110, SU 10 cycles are suspended during an SU 11 cycle, to transfer the next multiplier word into the L register. SU 10 cycles then resume and complete the multiplier alignment.

When the multiplier is in two storage words, VFL T4 trigger is set during SU 7 cycle as the fetch request is made for the second multiplier word. T4 trigger on during the SU 10 cycle in which the S pointer steps to zero gates the set of the gate J to AMTC trigger. The gate J to AMTC trigger is set to transfer the next multiplier word from the J register to the L register.

SU 11 Cycle: The SU 11 cycle transfers the next multiplier word from the J register through the main adder to the L register. The on status of the gate J to AMTC trigger gates the contents of the J register to the TC input of the main adder. Outgating the AMOB and release of the L register is conditioned by the on status of the J loaded trigger. Therefore, if an LCS is addressed, SU 11 waits several machine cycles until the J loaded trigger is set; otherwise, SU 11 is one cycle.

SU 12 Cycle: The SU 12 cycle transfers the multiplier from the K register through the RBL to the J register. At the same time, the low-order multiplicand word is transferred from the M register through the main adder and shifter to the K register. If VFL T1 trigger is on, the low-order multiplicand byte is in the 0 - 31 end of the M register. Therefore, the multiplicand word must be shifted right at least 32 bits (4 bytes) to right-align in the K register. The on status of VFL T1 trigger during SU 12 causes a right 32 shift to occur as the multiplicand is transferred from the M to the K register.

SU 12 cycle establishes gates to restore the T pointer to the starting byte address of the multiplicand; H register positions 21 - 23 are gated to the T latch and the T register released. In addition, gates are established to count the T pointer up and control byte alignment during PF 1 cycles.

PF 1 Cycle: PF 1 cycles are used to right-align the multiplicand in the K register if byte alignment is required.

If VFL T2 trigger is on, the multiplicand is already right-aligned in the K register when the first PF 1 cycle starts; PF 1 is then the only one cycle in which no function is performed. VFL T2 trigger is set during the SU 4 cycle if the low-order multiplicand digit is in byte 3 or 7 (T pointer equals 3 or 7) of the storage word. When the low-order multiplicand digit is in byte 7 of the storage word, the multiplicand is right-aligned. When the low-order digit is in byte 3 of the storage word, the right 32 shift during SU 12 right-aligns the multiplicand by transferring the data in byte 3 of the M register into byte 7 of the K register. Therefore, no further alignment is needed during PF 1.

When PF 1 is entered with VFL T2 trigger off, the multiplicand must be shifted right one or more bytes to right-align in the K register. Each PF 1 cycle shifts the multiplicand right 1 byte (8 bits) in

the K register. The shift is accomplished by gating the contents of the K register through the main adder and shifter with a right 8 shift gated, then back to the K register.

The T pointer controls the number of bytes the multiplicand is shifted and, therefore, the number of PF 1 cycles that occur. The T pointer is set to the starting byte address of the multiplicand, then stepped up one at the beginning of each PF 1 cycle. When the T pointer steps to 3 or to 7, the last PF 1 cycle is in progress and the set of PF 2 is gated.

PF 2 Cycle: The PF 2 cycle starts the final housekeeping functions of the set-up sequence by transferring the two right aligned operands, the low-order word of the multiplicand and the entire multiplier to their respective working registers. The multiplicand is transferred from the K register through the RBL to the J register. At the same time, the multiplier is transferred from the J register through the main adder to the L register.

PF 3 Cycle: During PF 3 cycle the sign digit is removed from the multiplicand and the multiplicand is shifted right one digit (4 bits) in the J register. This places the low-order multiplicand digit in positions 60 - 63 of the J register. The multiplicand is shifted by gating the contents of the J register to the RBL, and then back to the J register. Because this data path contains no right 4 shift gate, an effective right 4 shift is caused by shifting the output of the J register left 4 as it enters the RBL, then shifting the output of the RBL right 8 as it is gated back to the J register.

In addition, the PF 3 cycle prepares VFL controls to enter iteration cycles by resetting the S pointer and VFL T1 and T2 triggers. The VFL T7 trigger is set to provide product sign control gates during the first pass through the multiplier.

PF 4 Cycle: The PF 4 cycle is the last cycle of the decimal multiply set-up sequence. During the PF 4 cycle, the gate multiplier bus to DC line is active to transfer the low-order multiplicand digit into the DC. K and M registers are cleared to all zeros by gating AMOB to them. Because no inputs to the AM are active, the AMOB contains all zeros with correct parity.

During PF 4, the set of the SF and sequence D triggers are gated. When the SF trigger sets, the functions of the VFL sequence triggers change from set-up to store-fetch functions. VFL sequence D trigger is set to enter multiply iterations.

If the E interrupt trigger is on during PF 4 cycle, SF 3 is entered instead of sequence D. The E interrupt trigger on at this time indicates that the multiplicand sign is invalid or that the multiplier contains an invalid sign or digit. SF 3 starts a SF sequence to terminate the instruction and signal the interrupt condition.

Iteration Sequence

- Iteration Sequences performs the arithmetic of decimal multiply.
- VFL Sequencers IS 1, 2, 3, Seq A, B, C, D, and SF 12 are used.

The decimal-multiply iteration sequence multiplies the multiplier (Op 2) by the multiplicand (Op 1) to accumulate the product. The product is accumulated by adding or subtracting the multiplier over and over. The number of times the multiplier is added to or subtracted from the partial product depends on the value of each multiplicand digit.

When the iteration sequence starts, the multiplier is right-aligned in the L register, the low-order multiplicand word is right-aligned in the J register, the partial product field in the K and M registers is cleared to all zeros, and the low-order multiplicand digit is in the DC.

The iteration sequence then adds the multiplier to the partial product the number of times indicated by the digit in the DC. Each time the multiplier is added, iteration cycles start with the low-order multiplier byte and step through the multiplier and partial product fields adding one byte of each until the high-order multiplier byte is added; this constitutes one pass through the multiplier.

When a pass through the multiplier is complete, the DC is stepped up or down 1 and another pass is started. This process repeats until the DC steps down to 0 or up to 10. The next higher order multiplicand digit is then set into the DC, the multiplier is shifted left one digit, and the number of passes required to step the DC to 0 or 10 again are made through the multiplier. This process repeats until all multiplicand digits are processed.

After each pass through the multiplier the S pointer is restored to the value it contained at the beginning of the pass. Between passes the S pointer is reset to zero then stepped down one to 7; this causes each pass to start with the low-order multiplier byte. During each add cycle the T pointer gates the partial product byte added to the multiplier byte and selects the K register byte into which the new partial product byte is inserted. Therefore, after each pass through the multiplier, the T pointer is

restored to the value it contained at the beginning of the pass, except when the next pass starts with a new multiplicand byte. In this case, the Y counter is stepped down one and a new starting byte address for the T pointer is computed by gating Y to the AA and computing storage address B1 + D1 + Y. This address is then set into the H register.

The low-order multiplier digit must always align with the multiplicand digit in process throughout the execution sequence. Multiplicand digits are used one at a time from the J register, starting with the low-order digit and progressing through the multiplicand to the high order digit. Therefore, as each multiplicand digit is used, the multiplier must be shifted left one digit to align with the next multiplicand digit.

The multiplier is right-aligned in the L register during the set-up sequence; this alignment places the multiplier sign digit in the low-order digit position of the L register. Thus, in byte 7 of the L register, the low-order multiplier digit occupies the high-order digit of the byte and the multiplier sign occupies the low-order digit of the byte. With this alignment the low-order multiplier digit aligns with the low-order multiplicand-product digit; this is the high-order digit of the low-order multiplicand-product byte. The low-order multiplicand digit is processed and the first partial-product accumulated with this initial alignment.

Before the next multiplicand digit can be processed, the multiplier must be shifted left one digit. To do this, the multiplier is shifted right one digit (4 bits) in the L register (sign digit is shifted out), and the starting address of the T pointer is shifted left (reduced) one byte (8 bits). This places the low-order multiplier digit in alignment with the low-order digit of a partial-product byte (the correct alignment to process the low-order digit of a multiplicand byte). The multiplicand digit is then processed by making the required number of passes through the multiplier to step the DC to zero or 10. The high-order digit of the multiplicand byte is then set into the DC and processed. To align the low-order multiplier digit with the high-order digit of a multiplicand byte, the multiplier is shifted left one digit in the L register without changing the starting address of the T pointer.

As successive multiplicand digits are processed, the multiplier in the L register is alternately shifted right and left one digit. Data gates and sequence controls used to process the low-order digit of a multiplicand byte differ from those used to process the high-order digit of the byte. Therefore, odd cycles and even cycles are used to control the multiplier shifts and data gates to process each digit of a multiplicand byte. Odd cycles are cycles in which the multiplier is shifted left one digit in the L register and the high-order digit of a multiplicand byte is processed. Even cycles are cycles that occur when the multiplier is in the rightmost positions of the L register and the low-order digit of a multiplicand byte is being processed.

Odd and even cycle gating is controlled by the SC. The SC is reset to zero during the set-up sequence, then stepped up one as each multiplicand digit is readied for processing. Thus, odd or even cycle functions are gated by the odd or even numeric value contained in the SC.

VFL sequencers IS 1, IS 2, and IS 3 are used to perform the decimal arithmetic of the multiply instruction, while VFL sequencers A, B, C, D, and SF 12 are used to perform housekeeping functions such as counter updating, multiplier and multiplicand shifting, and so on. See Figures 66, 67, and 6475.

IS 1: IS 1 is the first sequencer in every pass through the multiplier when adding to the partial product. IS 1 controls the gates to AD for sign control and hot 1 for substraction.

IS 2: IS 2 is the sequencer after IS 1 during which K and L bytes are added. It loops on itself until a word boundary is crossed or the multiplier field is exhausted.

IS 3: IS 3 is the sequencer used at the end of a pass to propagate a carry of the partial product into the next byte. The first pass during even cycles and every odd pass through the multiplier is terminated by an IS 3 cycle. Although the IS 3 cycle is not needed to propagate a carry at the end of an even pass, it is needed to complement the next higher-order partial product byte when subtracting (Figures 64 and 65).

IS 3 is not needed during even cycles because the multiplier is shifted right 1 digit. This leaves a high-order digit to collect the carries.

SF 12: SF 12 is used to swap K and M when a word boundary is crossed during a pass through the multiplier.

Seq A: Seq A is the first sequence following an addition of the multiplier. It resets S, gates L2 to Z and gates H (B1 + D1 + Y) to T. If a word boundary was crossed during the addition, K and M will be swapped back at this time. DC is stepped down (up when subtracting) during Seq A and is decoded to determine if the digit multiply is complete. If it is, the next multiplicand digit is set to DC. If

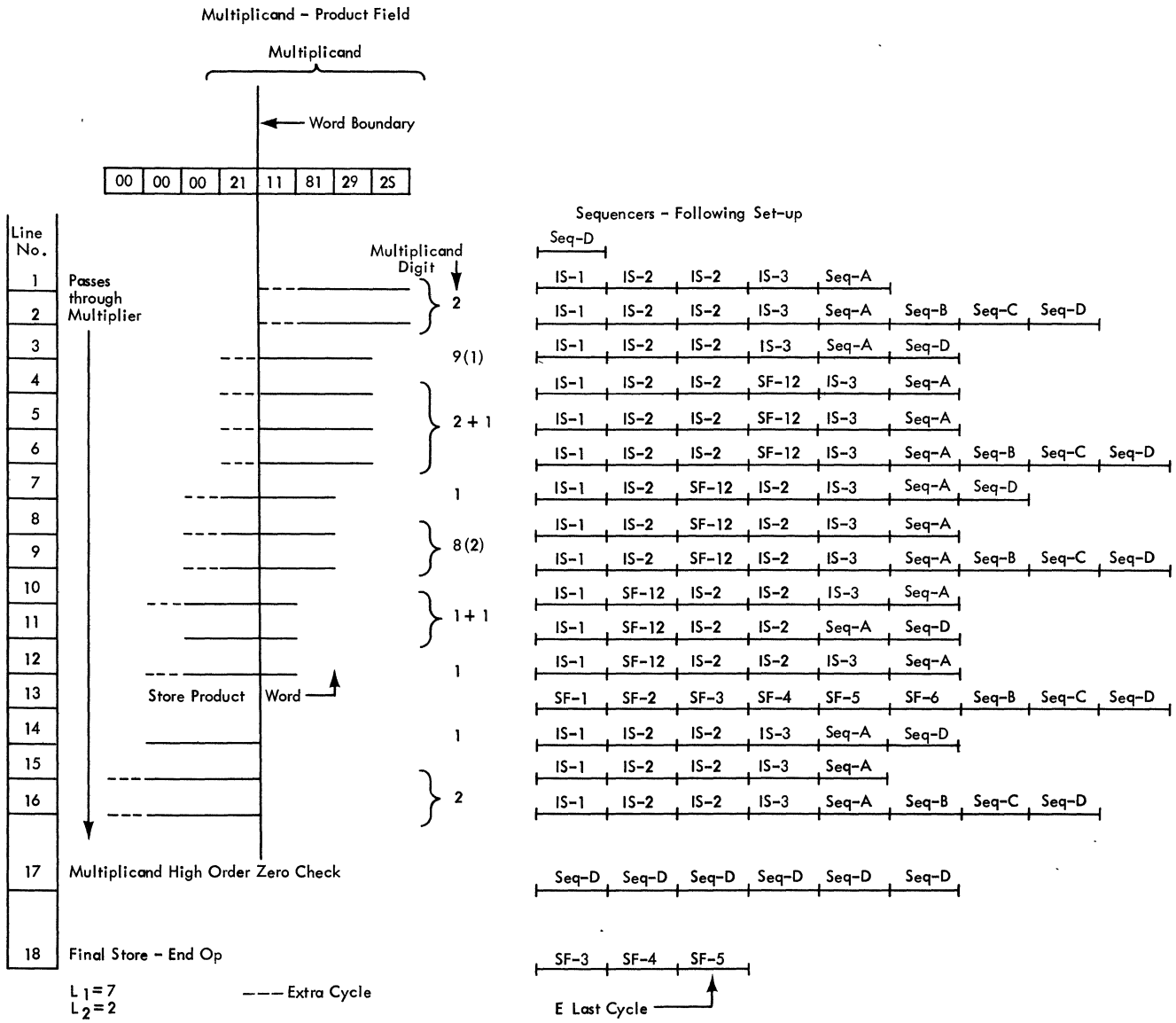
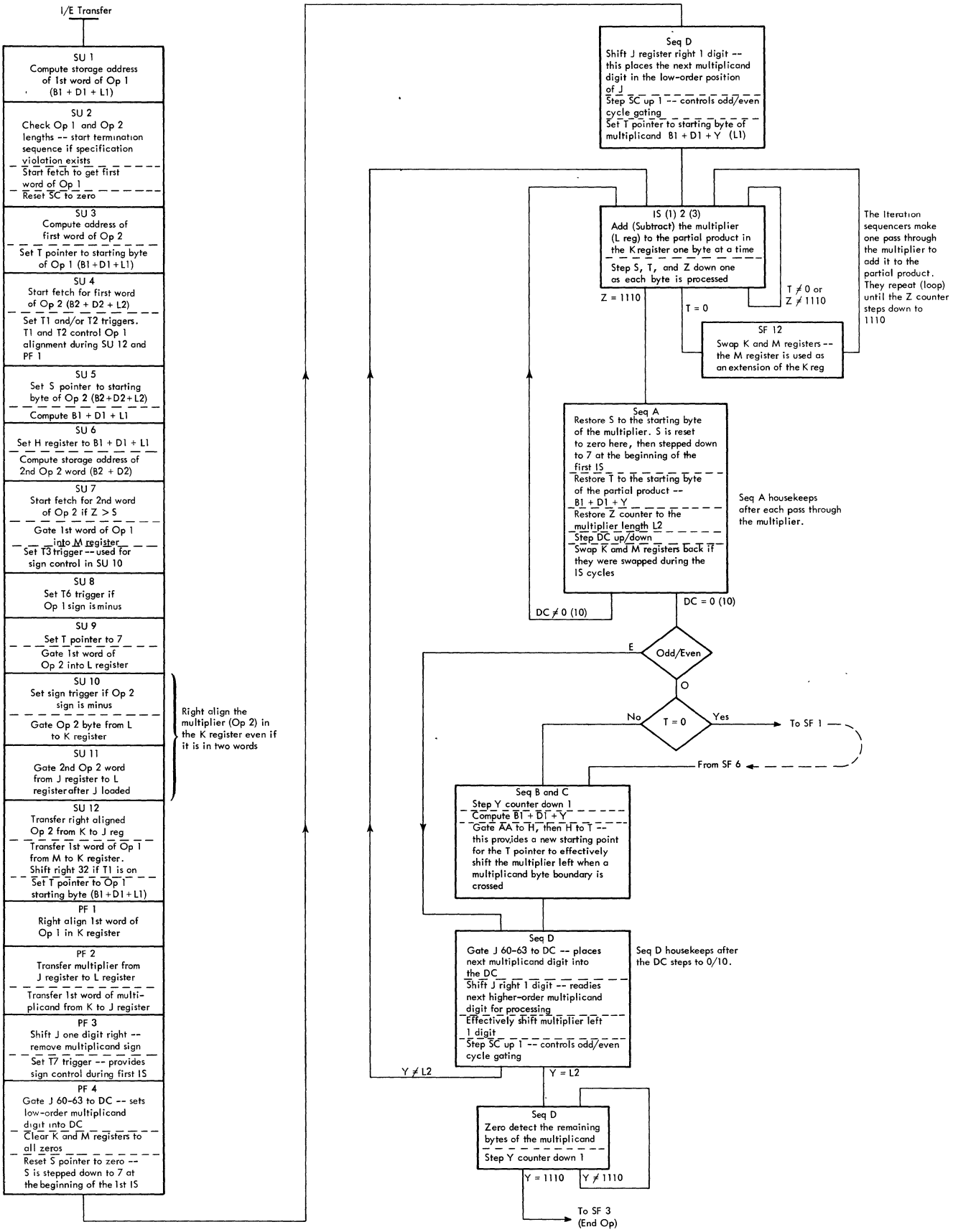


FIGURE 66. MULTIPLY ITERATIONS--EXAMPLE



● FIGURE 67. DECIMAL MULTIPLY (MP) SIMPLIFIED EXECUTION SEQUENCE--2075

not, another addition (subtraction) must be made so IS 1 is set. If the digit multiply is complete, the next even cycle sequence is Seq D. The next odd cycle sequence is Seq B.

Seq B, Seq C: When the odd cycle digit multiply is complete, the new Y value must be added to B1 + D1. Seq B and Seq C gate Y to AA and release H.

Seq D: Seq D shifts L right 4 following odd cycles, and left 4 following even cycles. The shift counter is incremented during Seq D. Seq D gates H to T to get the new T starting point if Y was stepped and added to B1 + D1. J is shifted right 4 to position the next multiplicand digit.

Store-Fetch Sequence -- Decimal Multiply

The store-fetch sequence for decimal multiply (Figure 6476) has two functions. First a fetch is requested for the next multiplicand word; then the completed portion of the product is stored from K register.

During a store-fetch, the Y counter tells how much of the multiplicand remains to be processed. Y is, therefore, used to compute the address of the words to be stored and fetched. If $Y > 7$, B1 + D1 + 1 is fetched and B1 + D1 + 2 is stored. If $Y \leq 7$, B1 + D1 is fetched and B1 + D1 + 1 is stored.

FIXED SEQUENCE VFL INSTRUCTIONS

- Use VFL data paths.
- Controlled by FXP sequencers.

Fixed sequence VFL instructions are those RX and SI instructions that, in general, handle one byte of data controlled by the FXP sequencers.

The following text and flow charts explain how these instructions are executed in System/360 Model 75.

Insert Character (IC)

The insert character instruction (Figure 6468) removes a data byte from the storage location specified by X2 + B2 + D2 and places it into the low-order byte of the general register specified by R1. The data byte may contain any bit configuration and may be located at any byte addressable location in storage. The data contained in the general register specified by R1 remains unchanged, except the inserted data byte.

Execution of the insert character instruction fetches the storage word to the L register and transfers the contents of the general register to the K register, positions 0-31. The selected byte of the L register is then gated through the decimal adder into byte 3, positions 24-31, of the K register. The contents of K register, positions 0-31, are then returned to the general register to complete the instruction.

E unit functions of the IC instruction start with the I/E transfer. Three E unit sequencers are used; first FXP, halfword logical (Hwd Log) and halfword add (Hwd Add). ELC is the put-away cycle and terminates the instruction.

Prior to the I/E transfer, the I unit computes the X2 + B2 + D2 storage address, sets it into SAR and H, and starts the fetch to get the storage word that contains the Op 2 data byte. In addition, the contents of the general register specified by R1 are gated through the RBL to M register positions 0-31.

The I/E transfer occurs as soon as the accept to the fetch request is received and starts execution in first FXP sequence. The contents of the general register gated through RBL is set into M register, positions 0-31, at the beginning of first FXP. First FXP cycles repeat until the Op 2 word arrives from storage and the J loaded latch is set. The byte address contained in H register, positions 21-23, indicate the Op 2 byte to be used. This byte address is set into the S pointer during first FXP sequence to control byte gating later. Because first FXP spans more than one cycle, and because the I unit may set a different address into the H register after the first E unit cycle, VFL T5 trigger is set for the first E cycle and T5 latch gates H 21-23 to the S and T pointers. VFL T5 insures that the byte address in H is not lost.

During first FXP, after J is loaded, data gates are established to transfer the operand 2 word from J register to L register through the main adder.

Hwd Log cycle follows first FXP. The operand 2 word is set into the L register at the beginning of Hwd Log cycle. During Hwd Log data gates are established to transfer the contents of M register (general register data) to K register through the main adder.

Hwd Add follows Hwd Log. Data from the M register is set into K at the beginning of Hwd Add. During Hwd Add the Op 2 data byte in the L register is selected and gated to the K register to be inserted with the other general register data. All bytes of the L register are inputs to the RBG. The contents of the S pointer control the RBG and gates the correct byte in the L register to the decimal adder. The decimal adder is gated binary true and provides a data path for the Op 2 byte. Decimal adder latches are gated to K. Byte gating into K register is normally controlled by the T pointer and T IN decode circuits, however, insert character instruction al-

ways inserts byte 3. Therefore, the contents of the T pointer is ignored and T IN decode is forced to 3 to insert the Op 2 bytes into K24-31.

ELC follows Hwd Add to restore the data to the general register (Rl) and terminate the instruction. The Op 2 byte is set into K register at the beginning of ELC. Then, the contents of K0-31 are transferred to the general register specified by Rl, and the instruction terminated.

Store Character (STC)

The store character (STC) instruction (Figure 6487) transfers the low-order, least significant, byte in the general register specified by Rl, to storage at the X2 + B2 + D2 address.

The K register is the only E unit register that gates data to the SBI enroute to storage. Therefore, the Op 1 byte is transferred from the general register to the K register, then to storage. Because the E unit contains no direct data path from a general register to K, intermediate data paths are used to transfer the Op 1 byte into K as shown in Figure 6487.

The E time functions of the store character instruction start with the first FXP cycle that follows the I/E transfer. Prior to the I/E transfer, however, the contents of the general register (R1) is gated through the RBL to M register, positions 0-31. In addition, the computed storage address, X2 + B2 + D2, is gated to the H register. The storage address and Op 1 data is set into the H and M registers at the beginning of the first FXP cycle.

During the first FXP cycle, the contents of the M register are gated through the main adder to the K register. Thus, the low-order byte, positions 24-31, of the general register is transferred to byte 3, positions 24-31, in the K register. At the same time, the three low-order storage address bits (the byte address) are transferred from H register, positions 21-23, to the S and T latches. S and T latches are then gated to the S and T registers.

The byte in the K register must be stored; therefore, VFL store request is set during first FXP and the contents of the T latch are gated to set the mark register.

Hwd Log cycle follows first FXP. At the beginning of the Hwd Log cycle, the data on AMOB is set into the K register; the S, T, and mark registers are set.

The Op 1 byte is in byte 3 location of K register. This may not be the byte location specified by the X2 + B2 + D2 address. Therefore, the Hwd Log cycle is used to correctly position the Op 1 byte in K. K register byte 3 is gated through the LBG and the decimal adder and back to the K register byte indicated by the T pointer.

Store sequence follows the Hwd Log cycle. If storage is busy and the store request initiated during the first FXP cycle must wait, the store sequence waits for the accept. Thereafter, ELC terminates the instruction.

AND, OR, and Exclusive OR

The three fixed sequence VFL logical connective instructions, AND-NI, OR-OI, and Exclusive OR-XI, are presented as a group because all use the same E unit controls and data paths except gating control to the AOE. See Figure 6488.

AND (NI)

When two operands are combined by the AND-NI instruction, they are matched bit for bit in the AOE. If corresponding bits are 1, the result bit is 1. If either bit is 0, the result bit is 0. For example, if the logical AND of the I2 byte and the B1 + D1 storage byte is performed, assume:

I2 byte	0101	1011
Storage byte (before)	0111	0110
Storage byte (after)	1101	0010

OR (OI)

When two operands are combined by the OR instruction, they are matched bit by bit in the AOE. If either of the corresponding bits is 1, the result bit is 1. If both bits are 0, the result bit is 0. For example, if the logical OR of the I2 byte and the B1 + D1 storage byte is performed, assume:

I2 byte	0101	1011
Storage byte (before)	0111	0110
Storage byte (after)	0111	1111

Exclusive OR (XI)

When two operands are combined by the Exclusive OR instruction, they are matched bit for bit through the AOE. If the corresponding bits match, either both 0 or both 1 bits, the result bit is 0; if they differ, the result bit is 1. For example, if the exclusive OR of the I2 instruction byte and the B1 + D1 storage byte is performed, assume:

I2 byte	0101	1011
Storage byte (before)	0111	0110
Storage byte (after)	0010	1101

The execution of the AND, OR, or Exclusive OR instruction starts with the I/E transfer. Prior to the I/E transfer, the I unit computed the B1 + D1 storage address and started the fetch to get the Op 1

word from storage. Immediate data in the I2 instruction field, IOP 8-15 is gated into the YZ counters. After the accept is received from the fetch request, the I/E transfer occurs.

First FXP is the E unit sequencer used to start the execution of the logical connective instructions. During first FXP the byte address is transferred from H register positions 21-23, to S and T latches and the Op 1 word is gated from the J register through the main adder to the L register as soon as it is received from storage. First FXP may span several cycles waiting for J loaded latch to be set. To insure that the byte address is set into the S and T pointers during the first E cycle, VFL T5 trigger is also set with I/E transfer. VFL T5 latch then gates H 21-23 to S and T latches.

Because the result byte from the AOE replaces the Op 1 byte in storage and must be stored, the VFL store request trigger is set during first FXP cycle after J loaded latch is set. The contents of the T latch is also gated to set the mark register when the store request is accepted.

Hwd Log cycle follows first FXP during which the two operands are gated through the AOE and the logical connective function is performed.

Op 1 byte is gated from the L register through the RBG to one AOE input. Op 2 byte is gated from the YZ counters to the other AOE input.

All 64 data bits of the L register are inputs to the RBG. The eight data bits of the byte selected by the S pointer become the output of the right digit gate and input to the right side of the AOE. The YZ counters contain the Op 2 byte; Y counter contains the high-order, most significant, 4 bits (HOD), and are inputs to positions 0-3 of left side of AOE. Z counter contains the low-order, least significant, 4 bits (LOD), and are inputs to positions 4-7 of left side of AOE.

The function of the AOE depends on the control gating. Two control gates, gate AND or gate exclusive OR, determine which of the three functions the AOE performs, AND, exclusive OR, or OR. If neither the gate AND nor the gate exclusive OR is active, the AOE performs the OR function. The AOE function gates are controlled by the instruction being executed.

The AOE latches are set to the result byte of the AOE. During latch time of the Hwd Log cycle the result byte is gated to the K register and to zero decode. The AOE result enters the K register byte selected by the contents of the T latch and T in decode circuits.

Store sequence follows Hwd Log cycle to store the result byte in K. The store request is initiated during first FXP sequence, and store becomes the wait sequence for the accept. When accept arrives, the store sequence ends and ELC starts.

ELC interrogates the on or off condition of the zero decode trigger and sets the condition register as shown in Figure 6488. ELC terminates the instruction:

Compare Logical (CLI)

The compare logical (CLI) instruction (Figure 6489) compares the immediate data byte, I2 field of the instruction, with the storage byte at the B1 + D1 storage address. The condition register is set to indicate the result of the comparison.

Prior to the I/E transfer, the I unit has computed the B1 + D1 storage address and initiated the fetch to get the Op 1 storage word. In addition, the immediate data, I2 field in IOP 8-15, is transferred to the YZ counters. The I/E transfer occurs as soon as an accept to the fetch request is received.

The I/E transfer releases the I unit to proceed to the next instruction while the execution of the CLI instruction is controlled by the E unit.

First FXP sequencer starts the execution of the CLI instruction. During the first FXP sequence, the byte address (H 21-23) is transferred to the S pointer, the data in YZ counters are transferred to the K register, and the Op 1 word is transferred to the L register.

Several machine cycles may occur before the word being fetched from storage is set into the J register. Therefore, first FXP cycles repeat until the J loaded latch is set.

J loaded latch enables the set of the first FXP latch and the continuation of the execution sequence. The byte address is normally gated from H register positions 21-23 to the S and T pointers by the first FXP latch. However, because first FXP sequence may span several cycles, and the I unit has proceeded to the next instruction, the address in the H register may be lost after the first E cycle. To insure that the byte address is set into the S and T pointer during the first E cycle, VFL T5 trigger is set for this first cycle and T5 latch gates the byte address from H register positions 21-23 to S and T pointer.

As soon as the Op 1 word arrives from storage into the J register it is transferred through the main adder to the L register. At the same time, the I2 data in the YZ counters are gated through the AOE to be set into byte 3 of the K register.

Hwd Log sequence follows first FXP during which the two operands are compared. The comparison is accomplished by subtracting Op 1 from Op 2 in the decimal adder.

Both operands are set into the K and L registers at the beginning of the Hwd Log cycle. The byte address is set from the S latch into the S register

at the same time. Both operands are then gated to the decimal adder, Op 1 through the RBG and Op 2 through the LBG. All bytes of the L register are gated into the RBG and the byte selected by the S pointer is the one gated to the decimal adder.

The Op 2 byte is gated from the K register to the decimal adder through the LBG. All bytes of the K register are inputs to the LBG. Normally the numeric value in the T pointer and T out decoding determine the K byte that is gated from the LBG to the decimal adder. However, because the immediate data was set into byte 3 of K during first FXP sequence, T decode out is forced to 3 during the Hwd Log cycle to gate K byte 3 through the LBG to the decimal adder.

Op 2 enters the left side of the decimal adder in true form. Op 1 enters the right side of the decimal adder through the TC +6 gate with complement gating. Op 1 is complemented to 2's complement and added to Op 2. The sum of the complement addition appears at the output of the decimal adder and represents the amount the operands differ.

The latched output of the decimal adder is gated to the bus to K register to be zero checked, however, K is not released and, therefore, the sum does not enter K.

ELC follows the Hwd Log cycle during which the condition register is set to indicate the result of the comparison as shown in Figure 6489. ELC terminates the instruction.

Move (MVI)

The MVI instruction (Figure 6490) moves the immediate operand (byte from the instruction stream) to the storage location specified by B1 + D1.

Prior to the first E cycle, the immediate operand is transferred from IOP 8-15 to the YZ counter and the store address is set into the SAR and H registers.

First FXP sequencer is set with the I/E transfer. During the first FXP cycle, the byte address is transferred from H register positions 21-23 to the T pointer to select the proper byte in the K register and set the correct mark bit. Because the MVI instruction is a store operation, the VFL store request trigger is set during the first FXP cycle.

Hwd Log sequence follows the first FXP cycle. The mark bit indicated by the T pointer is set at the beginning of the Hwd Log cycle. During the Hwd Log cycle, the immediate operand is gated from the YZ counter through the AOE to the K register byte selected by the T pointer. The YZ data is gated to the AOE during the early part of the Hwd cycle and appears at the AOE latch output almost immediately. From the AOE latch, the immediate operand is gated into the K register byte selected by the T pointer. Near the end of the Hwd Log cycle, the AOE latches are locked to retain the data set into them, and remain locked until the data are set into the K register at the beginning of the next cycle.

The store sequence follows the Hwd Log cycle. If storage is busy, store cycles repeat until accept arrives. If storage is not busy when the VFL storage request is made, then the accept will arrive during the Hwd Log cycle and the first store cycle is also the ELC. ELC terminates the instruction.

Set System Mask (SSM)

The SSM instruction (Figure 6491) removes a data byte from storage and places it in positions 0-7 of the PSW.

The SSM instruction conforms to the SI instruction format, however, the I2 field is ignored. The data byte used for the system mask is located in storage at the B1 + D1 storage address.

The E unit and the I unit are both involved in the execution of the SSM instruction. E unit sequencers select and gate the desired byte from the storage word to the I unit. IE sequencers then control the gating and setting of the mask byte into the PSW register.

E unit functions of the SSM instruction start with the I/E transfer. Prior to the I/E transfer, the I unit computes the B1 + D1 storage address, sets the address into SAR and H, and starts the fetch request to get the storage word that contains the mask byte. I/E transfer occurs after the fetch request has been accepted, and starts first FXP sequence.

The E unit remains in first FXP sequence, repeating cycles, until the B1 + D1 storage word is set into the J register and the J loaded latch is set. Because first FXP may repeat several cycles, VFL T5 trigger is set for the first E cycle to insure that the byte address is transferred from H register to the S and T pointers.

During first FXP, after J is loaded, the contents of the J register is gated through the main adder to the K register. AMOB is gated to K with first FXP latch and K is set when the following cycle starts. Decimal go signal is sent to the I unit to start the I execute sequence IE 1.

The last E unit cycle, ELC, and the IE 1 cycle follow first FXP and occur at the same time. ELC controls the E unit function of gating the mask byte from the K register through the LBG to the I unit where the IE 1 cycle gates it into the PSW. During ELC, all bytes of the K register are inputs to the LBG. The TD out trigger and the contents of the T register (the byte address) control the selection of the byte gated out of the LBG. The mask byte is routed to the I unit where the IE 1 cycle gates it into

the PSW register positions 0-7. IE 2 cycle follows IE 1. The mask byte is set into the PSW register at the beginning of the IE 2 cycle to complete the execution of the SSM instruction. IE 1 cycle follows IE 2. See I unit section for details of IE sequencing.

Test Under Mask (TM)

The TM instruction (Figure 6492) uses the immediate data byte, I2, as an eight-bit mask to select and test the state of the corresponding bits of the Op 1 (B1 + D1) data byte in storage. The 1 or 0 state of the selected bits is used to set the condition register.

A mask bit in the 1 state indicates the corresponding bit of the storage byte is selected. When the mask bit is 0, the corresponding storage bit is ignored; when all selected storage bits are 0, or when all mask bits are 0, both condition register bits, 34 and 35, are set to 0. When the selected bits of the storage byte are all 1's condition register bits 34 and 35 are set to 1's. When selected bits are mixed, 1's and 0's, the condition register is set to 01.

At the programmer's option, the mask byte may contain any configuration of bits ranging from all 0's to all 1's, depending on the program objectives. The following examples show some possible objectives, mask and storage byte relationship and the condition code setting.

Example 1.

Objective: Test bits 2, 3 and 7 of a storage byte for 1's. The mask byte contains 1's in positions 2, 3 and 7, and 0's in all other positions.

Bit positions	0	1	2	3	4	5	6	7
Mask byte	0	0	1	1	0	0	0	1
Storage byte	1	1	1	1	0	1	0	1

In this example, the storage byte contains a mixture of 1's and 0's; however, the selected positions 2, 3 and 7, contain all 1's. When selected bits are all 1's, the condition register is set to 11. The bit positions of the storage byte that correspond to 0 mask bits are ignored and have no effect on the condition code.

Example 2.

Objective: Test bits 2, 3, 4 and 6 of a storage byte for 1's.

Bit positions	0	1	2	3	4	5	6	7
Mask byte	0	0	1	1	1	0	1	0
Storage byte	1	1	1	1	0	1	0	1

In this example, the same storage byte as Example 1 is used. The mask byte is changed to test bits 2, 3, 4 and 6. Bits 2 and 3 of the storage byte are 1's, while bits 4 and 6 are 0's. The selected bits are a mixture of 1's and 0's; therefore, the condition register is set to 01.

If the mask bits are all 0's or the selected bits are all 0's, the condition register is set to 00.

Execution of the TM instruction AND's the bits of the mask byte with the corresponding bits of the storage byte in the AOE. Two output lines of the AOE, mask all 1's and mask all 0's, control the setting of the condition code. Figure 68 is a simplified positive logic diagram that shows the AND functions of the mask and storage bytes, and the two AOE output lines. When the mask and storage byte are AND'ed, either both, or neither, the mask all 1's or mask all 0's line is active, depending on the bit configuration of the two bytes:

	Active AOE Output	
	Mask all ones	Mask all zeros
Selected bits all zero		X
Mask bits all zero	X	X
Selected bits all ones	X	
Selected bits mixed		

Execution of the test under mask instruction fetches the operand 1 word (B1 + D1) from storage, AND's the selected byte with the I2 instruction field (mask byte) and sets the condition register. E unit execution starts with the I/E transfer. Prior to the I/E transfer, however, the I unit computes the B1 + D1 storage address and starts the fetch request to get the Op 1 word from storage. In addition, the I2 instruction field (mask byte) is set into the Y and Z counter. The I/E transfer occurs after the accept to the fetch request is received from the BCU.

First FXP sequence follows the I/E transfer and repeats cycles until the Op 1 word arrives from storage and is set into the J register. To insure that the byte address is transferred from the H register to the S and T pointers during the first E cycle, VFL T5 trigger is set and T5 latch gates H 21-23 to S and T.

During the first FXP cycle after J is loaded, the storage word in J register is gated through the main adder and to the L register. Data is set into the L register at the beginning of the next cycle.

Hwd Log sequence follows first FXP. During Hwd Log cycle the mask bits are used to test the bits of the Op 1 byte. All bytes of the Op 1 word in the L register are inputs to the RBG. Gate L with S trigger and the byte address in the S pointer se-

lect and gate the Op 1 byte through the RBG to the AOE where it is tested. At the same time, the mask byte in the YZ counters is gated to the other AOE input. The AND function to the AOE is gated to enable the mask bits to select and test bits of the Op 1 byte. The two AOE output lines, mask all 1's and mask all 0's indicate the 1 or 0 state of the bits selected by the mask byte. VFL T7 and T8 triggers are used to remember the status of the AOE output and to set the condition code during the next cycle. VFL T7 and T8 are used because the YZ counters are normally set to a new value during ELC. The chart in Figure 6492 indicates the set conditions for VFL T7 and T8 triggers and the condition code that results.

ELC sets condition register positions 34 and 35 and terminates the instruction.

Test and Set (TS)

This SI format instruction (Figure 6493) tests a single byte in main storage for a high-order bit and then sets the entire byte tested to all 1's. The byte tested, then set to 1's, is specified in the first operand address. The result of the test for a high-order 1 bit is recorded in the condition code. If the high-order bit in the selected byte is a 0, the condition code is set to 00. If the high-order bit is a 1, the condition code is set to 01.

The storage unit does a unique operation for the test and set instruction. The addressed storage word is fetched and set unaltered into the SBO latch register exactly as during a fetch operation. Unlike a normal fetch, however, the storage uses a mark-bit supplied by the CPU to designate a single byte to be changed in storage. The storage unit sets the designated byte to all 1's then regenerates the 72-bit word. Thus, the storage unit does a combination store and fetch.

To cause a test and set, the BCU does a normal CPU fetch but sends a test and set signal to the selected storage unit. No special gating is required for the mark-bit. The CPU sets a bit into the mark register; the mark register is gated to storage on any CPU operation. A unique mark register reset, however, is required for the test and set instruction. The mark register is reset after any CPU store operation and after the test and set instruction.

The test and set instruction is executed by setting a bit into the mark register by decoding bits 21-23 of the address set into the H register. SAR is set in parallel with H from the addressing adder and fetch request and test and set signals are sent to the BCU, and the return of J line brought up.

When J is loaded, its contents are gated through AMTC to the L register. The first fixed-point trigger is turned on and stays on until J is loaded. The S and T registers are set from H 21-23 during the first FXP cycle by VFL T5 trigger.

The high-order bit test is performed in the byte of L register selected by the S pointer. ELC controls the setting of the condition code and terminates the instruction.

The storage address protection unit is active on a test and set instruction. A SAP check causes the original word to be regenerated in storage and, instead of the fetched word, the storage unit delivers all zeros with good parity-bits to the SBO latch register. This protects the CPU from taking a machine check caused by a SAP error.

CONVERT INSTRUCTIONS

- Convert decimal data to binary.
- Convert binary data to decimal.

Input data to the System/360 may be in either of two formats, binary or binary-coded-decimal. Of all Systems/360 instructions, certain ones, such as fixed-point and floating-point, require that data be in binary format; other instructions, such as VFL decimal, require data in BCD format.

Because data may enter the system in either format, correct execution of an instruction may require that the data to be processed be converted to the correct format.

Two instructions, convert to binary and convert to decimal, provide the facility to convert data from one format to another.

The following text and flow charts explains how the System/360 Model 75 executes the convert to binary and convert to decimal instructions.

Convert to Decimal (CVD)

The CVD instruction converts the 32-bit binary word in the general register specified by R1 instruction field into BCD digits, and places them in storage at the X2 + B2 + D2 address. A storage word can contain 15 decimal digits plus a sign digit. The maximum decimal value that can be represented by 15 digits is far greater than the maximum value represented by 31 binary bits plus a sign bit, therefore an overflow cannot occur during conversions.

BCD digits are represented by 4 binary bits. The difference between BCD data and pure binary data is that a 4-bit BCD digit is allowed to contain only 10 different values, 0-9, whereas 4 bits used for pure binary data may contain 16 different values, 0-15. The values within the range of 0-9, 0000-1001, are

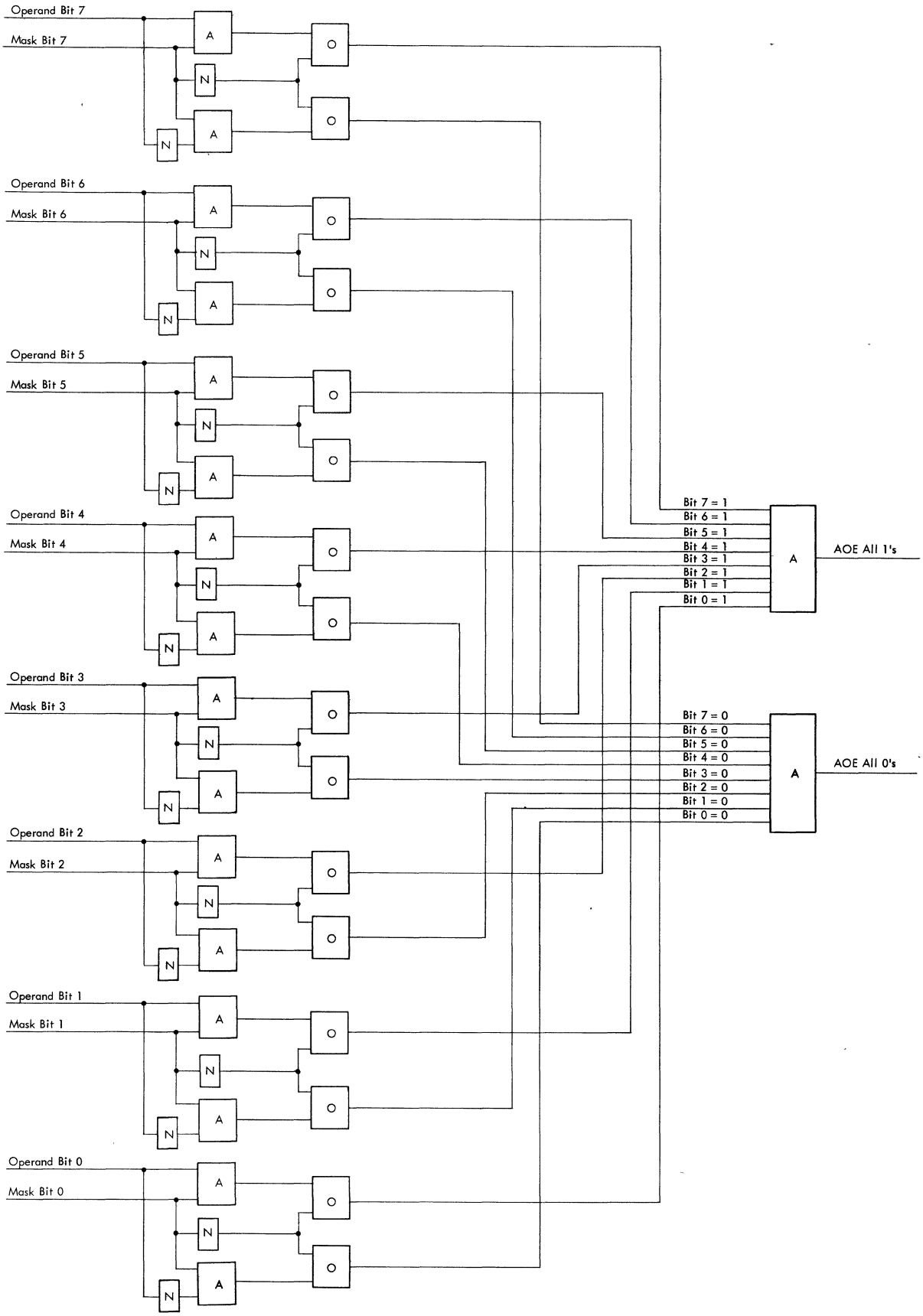


FIGURE 68. AOE MASK FUNCTION

the same in either binary or BCD coding. However, values greater than 9 are expressed differently in binary and BCD coding. For example, the numeric value of 12 is 1100 in binary and 0001 0010 in BCD.

Numeric values expressed in binary form represent the sum of the binary digits, or bits. The value of 12, for example, is the sum of the binary 8 and 4 bits.

Concept

When converting numbers from binary to BCD, the number is converted from the base 2 system to the base 10 system. Any number of any number system can be expressed as:

$N = An\ q^n + An\text{-}1^{n-1} + An\text{-}2q^{n-2}$ ----------

N = The number of any number system
An, An-1+An-2, etc., = non negative digits of the number
q = The base number of the system
n = The power (position) of the digit within the number

For example, the decimal number 985 can be expressed as:

$$985 = 9x10^2 + 8x10^1 + 5x10^0$$

or in binary as:

$$985 = 1x2^9+1x2^8+1x2^7+1x2^6+0x2^5+1x2^4+1x2^3+0x2^2+0x2^1+1x2^0$$

factored to:

$$985 = (((((((((\ 1x2+1)2+1)2+1)2+0)2+1)2+1)2+0)2+0)2+1)$$

The conversion of binary numbers to decimal numbers can be accomplished by the execution of the above equation, Figure 69.

The same equation is used when System/360 Model 75 executes the CVD instruction. Convert cycles convert the binary number to 4-bit BCD digits. One binary digit, or bit, is converted during each convert cycle, starting with the highest-order binary digit. Thereafter, the next lower-order binary digit is converted, and the cycles repeated until all binary digits are added to 2 times the partial result, which in the beginning is zero, and a new partial result developed. The 2x multiple of the partial result is produced by shifting left 1. Figure 70 shows a simplified version of how binary number 985 is converted to 985 in BCD. Ten cycles are shown, during which the partial result is expanded by multiplication and addition of each binary digit. When the last binary digit is added, the result is complete, in BCD digits.

A BCD digit occupies 4 bit positions of storage and may represent any decimal number from 0-9. The same 4 bit positions may also represent any binary number from 0-15. Numbers 0-9 are the same, whether in binary or BCD form; however, binary numbers greater than 9 (1001) become invalid BCD digits. For example, one more than 9 (1001) in BCD becomes 0001 0000, whereas in binary, one more than 9 (1001) becomes 1010, an invalid decimal digit. During the convert sequence, if a partial result digit is greater than 4, an invalid BCD digit is created when the shift left 1 occurs to produce a 2x multiple of the partial result. If, for example, the partial result is a 5 (0101) before the shift, then it becomes 1010 (binary 10) after the shift. Decimal correction circuits provide +6 correction to all BCD digit positions that contain a sum of 5 or greater before the X2 shift. The +6 correction forces a carry into the next higher-order BCD position. Figure 70 shows the +6 correction being used. At the end of cycle 3, the units order 4-bit position of the partial result contains a 7 (0111). When the 7 (0111) is shifted left 1, the binary sum becomes 14 (1110), and when the next binary digit is added, the binary sum becomes 15 (1111), however, +6 is also added. The +6 forces a carry into the adjacent 4-bit BCD position and provides the correct BCD digits for the sum of 15 (0001 0101). Decimal correction is provided for each of the decimal digits converted during the CVD instructions.

Execution

Execution of the CVD instruction starts during I unit sequencing when the contents of the general register specified by the R1 instruction field are gated through the RBL to M register. Also, 1 is gated to the input of SC position 2 to set 32 into the SC.

E unit execution of the CVD instruction, Figure 6494, starts with the first FXP sequencer. During first FXP cycle Op 1 is routed through the main adder to the K, L and M registers. If Op 1 is a negative number, it is in complement form when transferred to M from the RBL, and the R1 sign trigger is set. If the R1 sign trigger is on during first FXP, the AM complement and hot 1 triggers are on and Op 1 is complemented and changed to true form.

Op 1 may be any value from 0 to +2,147,483,647, or -2,147,483,648, the maximum value that can be represented by 31 binary bits; the magnitude of Op 1 is checked during first FXP by zero detection of the 4 high-order bit positions of M. If M 0-3 equals zero, normalize cycles follow first FXP. If M 0-3 does not equal zero, convert cycles are started.

Normalize Cycles: When M 0-3 equals zero, normalize cycles follow first FXP. At the beginning of the first normalize cycle, Op 1 is set into the K, L and M registers. K register positions 0-63 are zero detected, and if zero, indicating Op 1 has zero value, conversion is not necessary. IS 1 cycle is entered and the sign digit placed in the low-order position of K register.

When K0-63 is not equal to zero, normalize cycles repeat until Op 1 is digit normalized. During each normalize cycle, M 0-3 and M 0-11 are zero

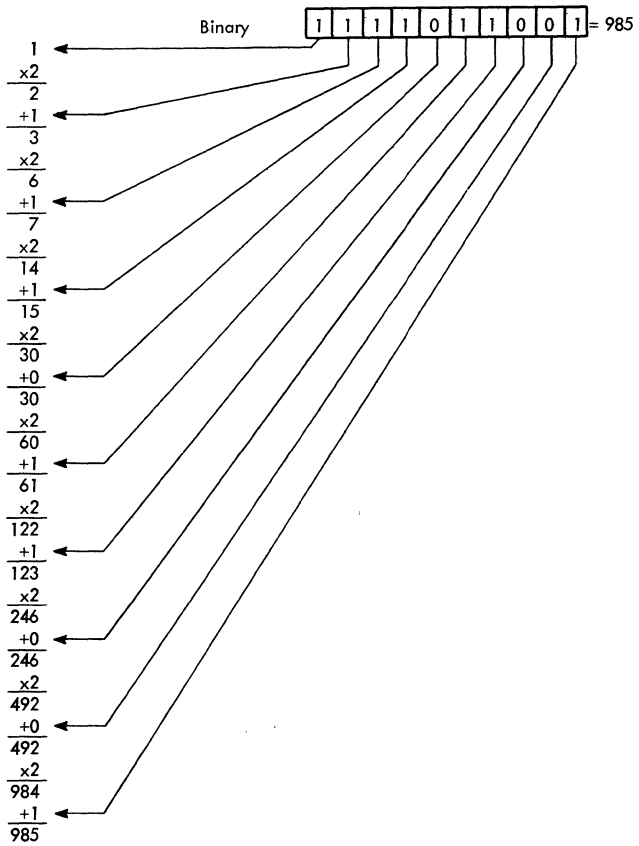


FIGURE 69. CONVERT BINARY TO DECIMAL

detected. M 0-63 is gated through the main adder, the shifter, and to AMOB, then to K, L and M registers. M 0-3 and M 0-11 zero detection controls the gates to the shifter. M 0-3=0 causes a L4 shift and M 0-11=0 causes a L8 shift. M 0-11 zero detection is a look-ahead feature; when M 0-11=0, the normalize cycle repeats. When M 0-11 is not equal to zero, the last normalize cycle is in progress and and last 4 or 8 zeros are being deleted from Op 1.

Initially the shift counter is set to the value of 32 to count the Op 1 bytes processed. During each normalize cycle, the SC is stepped down either 4 or 8 depending on the number of zeros shifted out of the Op 1 field.

During each normalize cycle, AMOB is gated to K, L and M registers. The last normalize cycle gates Op 1, left digit aligned, into the registers and sets the convert trigger.

Convert Cycles: Convert cycles start as soon as the convert trigger is set. During each convert cycle, an Op 1 binary bit is converted to BCD form, and the SC stepped down 1. When the SC = 0, all bits of Op 1 have been converted and the BCD number is in the K register.

At the beginning of the first convert cycle, the L register contains Op 1, left digit aligned in positions 0-32. The partial result is developed in L register positions 32-63. During each convert cycle, L 0-63 is gated L1 (x2) to AMTC, L 32-63 is gated through decimal correction to AM inputs 32-63; if SC is less than 24, then L28-31 is also gated through decimal correction to AM 28-31. The binary bit is gated from L0 to AM 63, where it is added to 2x the partial result and develops a new partial result. The new partial result is gated from AMOB to K and L registers 0-63. The shift counter steps down 1 at the beginning of each convert cycle, and when SC=0, the last binary digit is being added to 2x the partial product; SC=0 terminates the convert sequence and starts the IS 1 cycle to place the sign digit in K register.

Decimal correction circuits examine the binary sum of the bits in each 4-bit digit position of L 32 to 63 if SC > 24 or L 28-63 if SC< 24. Each 4-bit digit position of L register that contains a bit sum of 5 or greater causes +6 to be gated from decimal correction to AM inputs for that position. AM inputs from decimal correction are 0 for all 4-bit digit positions with a bit sum less than 5.

Each convert cycle shifts the contents of the K and L register 1-bit position to the left. Thus, a new Op 1 bit is gated from L0 to AMTC 63 and added to the X2 partial result each convert cycle.

Convert cycles repeat until the SC steps down to 0. SC=0 is detected during the last convert cycle

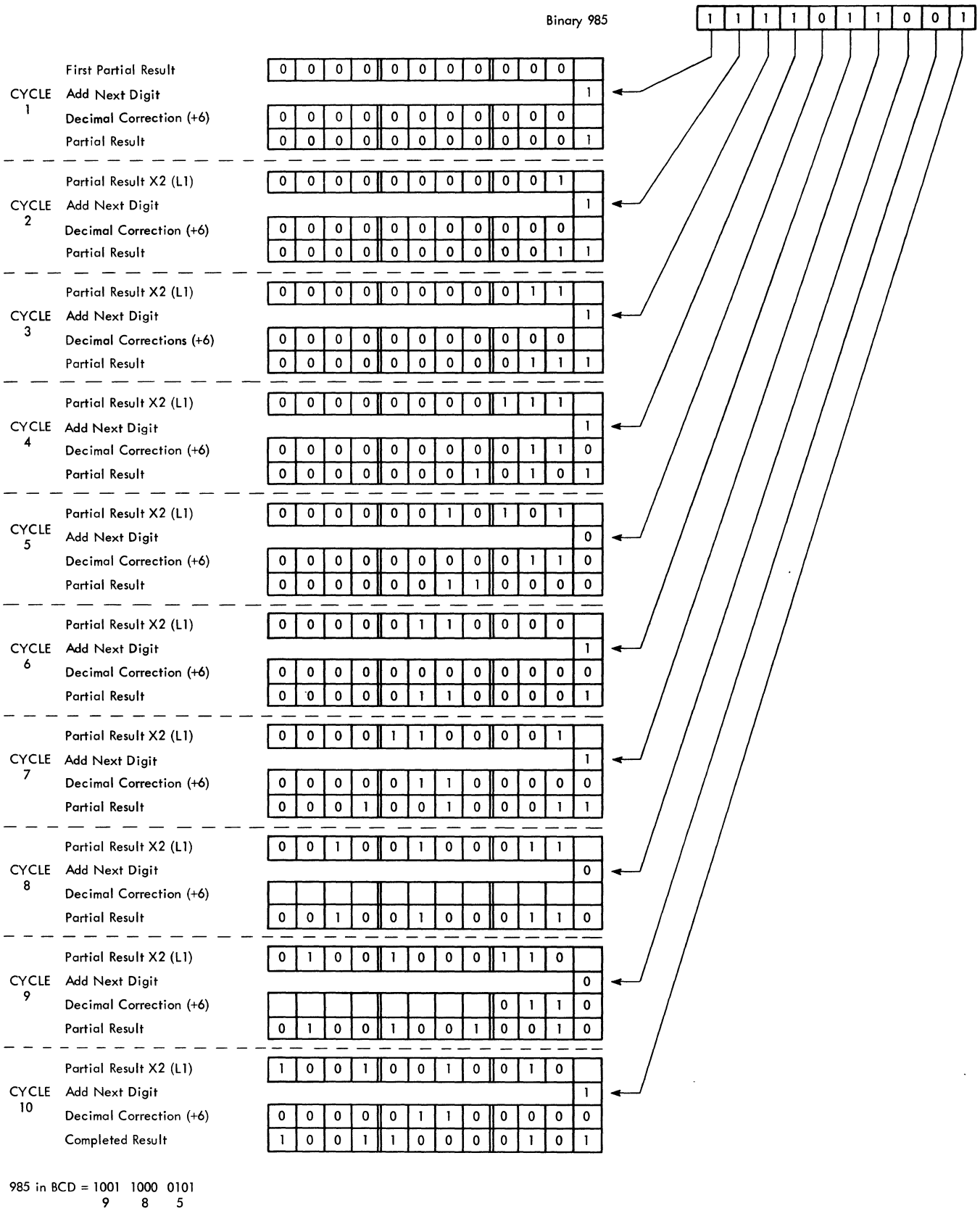


FIGURE 70. CONVERT BINARY TO BCD

and causes the convert trigger to be reset and the IS 1 trigger to be set. The last convert cycle gates the completed result into K, L and M registers, with a left shift 4 to make room for the decimal sign.

IS 1 Cycle: IS 1 cycle places the correct sign digit into the low-order digit position of K register. Because K register is byte addressable and not digit addressable, VFL circuits are used to insert the sign digit. The low-order byte (byte 7) of K is gated through the LBG, and the HOD gated to the left 0-3 input of the decimal adder. A plus sign is forced to the right LOD input (4-7) of TC +6 side of the decimal adder. If the R1 sign trigger is on, the sign digit is inverted and a minus sign gated through the decimal adder. The sign digit forced to the right LOD input and the HOD from the LBG are gated through the decimal adder and back to byte 7 of the K register.

The T pointer is used during IS 1 cycle to select the K byte. Initially, the T pointer is set to 0, then, just prior to IS 1, T is gated to step down 1. At the beginning of IS 1, T is then stepped from 0 to 7. J Register gates byte 7 through the LBG and the latch T controls the release of byte 7 into K from the AV latches.

The setting of the VFL store request and the store request and the store triggers is gated during IS 1 cycle, to store the completed result at the Op 2 address.

Store Cycle: The store cycle follows IS 1 and is the accept wait cycle for the store request made during IS 1. When accept arrives, ELC occurs and the operation ends.

Convert to Binary (CVB)

The CVB instruction converts the BCD data in the Op 2 storage word to binary data and places it in the general register specified by R1.

The 64 bit Op 2 (X2 + B2 + D2) storage word must conform to the packed decimal format, and may contain as many as 15 decimal digits plus the sign digit. The 15 decimal digits are converted to a 31-bit binary word plus the sign bit and stored in a general register. Decimal data is always true; however, the sign digit may be plus or minus. Decimal data is converted to plus binary data; then, if the decimal sign is minus, the binary data is changed to the 2's complement before it is set into the general register.

The maximum signed decimal number that can be contained in a 64-bit storage word is 10^{15}-1, however, the maximum number that can be converted and still be contained in a 32 bit register is +2,147,483,647 or -2,147,483,648. When the decimal number is outside this range, the low-order 32 binary bits are placed in the general register and a fixed-point divide interrupt occurs.

The convert to binary instruction converts one decimal digit at a time, starting with the high-order digit and converts, a digit at a time, to the low-order digit. Figure 71 shows an example of convert to binary. The partial result, which at the beginning is zero, is multiplied by 10 and the first digit added to it. The 10X multiple of the partial result is the sum of X2 and X8 multiple. X2 and X8 multiples are created by shifting the partial result left 1 position (L1) for the X2, and left 3 positions (L3) for the X8. The partial result, after the first decimal digit is added, is multiplied by 10 and the next digit added. This procedure is continued until the BCD number is exhausted, at which time the result bits represent, in binary sum, all the decimal digits.

Decimal digits are converted to binary in the main adder. The decimal digit is gated to the AM from the high-order digit position (0-3) of the K register and added to the partial result gated to the AM from the L register (Figure 72). To effectively multiply the partial result by 10, positions 0 to 63 of L register are gated with a L1 (X2) shift to AMTC positions 0-62, and L register positions 28-63 are gated L3 (X8) to AM positions 25-60. AM positions 61, 62 and 63 and AMTC 63 are used to enter the decimal digit from the K register. Because BCD digits are represented by 4-bit positions and only the 3 low-order positions of the AM are available, modified gating is used to enter the 4-bit BCD digit. Main adder positions AM 61, AM 62, and AMTC 63 are used to enter the BCD digit if its value is 7 (0111) or less. Figure 72 shows the relationship of the BCD bits from the K register to AM inputs. K register position 03 (BCD 1 bit) is gated to AMTC 63; K register positions 01 and 02 (BCD bits 4 and 2) enter AM 61 and 62. K register position 0 (BCD 8 bit) causes a carry in to position 63. If the BCD digit from K register is 8 (1000), 1's are forced to AM 61, 62 and 63 and a forced carry into AM 63 causes a carry into the binary 8 position (60) of the AM. When the BCD digit is 9 (1001), 8 (1000) is entered as above and the 1 (0001), from K03, enters AMTC 63.

During each convert cycle, the BCD digit in the high-order position (0-3) of K register is added to 10X the partial result from the L register, Figure 72. The new partial result is returned to the L register. At the same time, the next BCD digit is gated from the K register through the RBG, and decimal adder into the high order of K, ready for the next convert cycle. The K byte gated through the RBG is controlled by the setting of the S pointer. Initially, the S pointer is set to 0 and stepped up 1 on alternate convert cycles.

During the convert cycle that the first BCD digit is converted, the S pointer is 0. K byte 0 is, therefore, gated through the RBG. The digits are crossed causing the LOD of K byte 0 to gate through the HOD position of the decimal adder and into the high-order position (0-3) of the K register.

The next cycle converts the LOD of K byte 0, and because the S pointer is stepped up 1 at the beginning of this cycle, byte 1 from K register is gated through the RBG. The digits are straight gated through the digit gate to the decimal adder; the HOD of K byte 1 is gated through the decimal adder into K register positions 0-3, ready for the next cycle.

The CVB instruction proceeds in the above manner until all digits in the K register have been converted.

The S pointer counts and controls the gating of each Op 2 byte from the K register as the high- and low-order digits of the byte are converted. In addition, the shift counter is used to count the digits converted. Initially, the shift counter is set to 16 and counted down 1 each convert cycle. When the S pointer equals 7 and the shift counter equals 1, the last Op 2 digit is being converted. The CVB instruction then transfers the 32-bit word to the general register specified by R1 and the instruction is terminated.

Execution controls and data flow for the CVB instruction are shown in Figure 6495. The inset at the bottom of Figure 6495 shows the relationship of the counter values and digits converted during each convert cycle.

CVB Execution

E unit execution of the CVB instruction starts with the first FXP sequence, set with the I/E transfer. Prior to the I/E transfer, the I unit computes the Op 2 (X2 + B2 + D2) address and starts the fetch to get the Op 2 word from storage. In addition, the shift counter is set to 16 by forcing 1 into SC position 3. BCU accept causes the I/E transfer to occur and set the first FXP sequencer.

The E unit waits in the first FXP sequence until the Op 2 word arrives from storage and is set into the J register. When the J loaded latch is set, the Op 2 word is gated through the main adder to the K register. The S pointer is reset to zero, and the convert and IS 1 triggers are set to control the CVB instruction. The convert trigger remains on throughout all the convert cycles; the convert trigger controls the gating of the partial result from the L register and the digit from the high-order position (0-3) of K register to the main adder, and the partial result to L register. IS 1 trigger is on during the first cycle the convert trigger is on to set IS 2. IS 2 and IS 3 sequencers are used thereafter to control VFL circuits and gates to position each succeeding Op 2 digit into the high-order of K register.

Convert/IS 2 cycle and convert/IS 3 cycles alternate and repeat until all decimal digits in the K register are converted. The HOD of each K byte is converted during IS 2 cycles and the LOD of each K byte is converted during IS 3 cycles.

Convert/IS 2: When the first IS 2 cycle starts, data gates to the main adder from the L register and from positions 0-3 of the K register are established and remain active during each succeeding cycle until all Op 2 digits are converted. Data gates from the L register provide the X10 partial result to which the BCD digit from K 0-3 is added. During IS 2 cycles, the HOD of the K byte is converted; at the same time, the same K byte is gated through the RBG, the digits are crossed, and the LOD gated through the HOD positions of the decimal adder. The decimal adder output is then gated into K register positions 0-7, placing the HOD in position 0-3 and the HOD in 4-7; this places the LOD of the byte in position to be converted during the next cycle.

Convert/IS 3: The IS 3 cycle follows IS 2. During IS 3 cycles, the LOD of the K bytes is converted and the HOD of the next K bytes is transferred to positions 0-3 of the K register.

The S pointer is stepped up 1 at the beginning of each IS 3 cycle. The next K byte is then gated through the RBG, straight gated to the decimal adder and placed in K 0-7; this places the HOD of the new K byte into position to be converted during the next cycle.

The convert sequence returns to IS 2 cycle unless the S pointer stepped to 7 at the beginning of IS 3. S = 7 during IS 3 indicates that only one digit remains to be converted; the last K byte, byte 7, is gated through the decimal adder and the byte is set in K 0-7. The HOD in K 0-3 is the next digit to be converted; however, K 4-7 is the sign digit of Op 2, and is not converted. During IS 3, when the sign digit is gated to the decimal adder, the E unit sign trigger is set if the decimal sign is minus.

Last Convert Cycles: When S = 7, two convert cycles follow IS 3. The first converts the last decimal digit to binary; the second transfers and aligns the completed result into the K register ready to be set into the general register during the PA cycle.

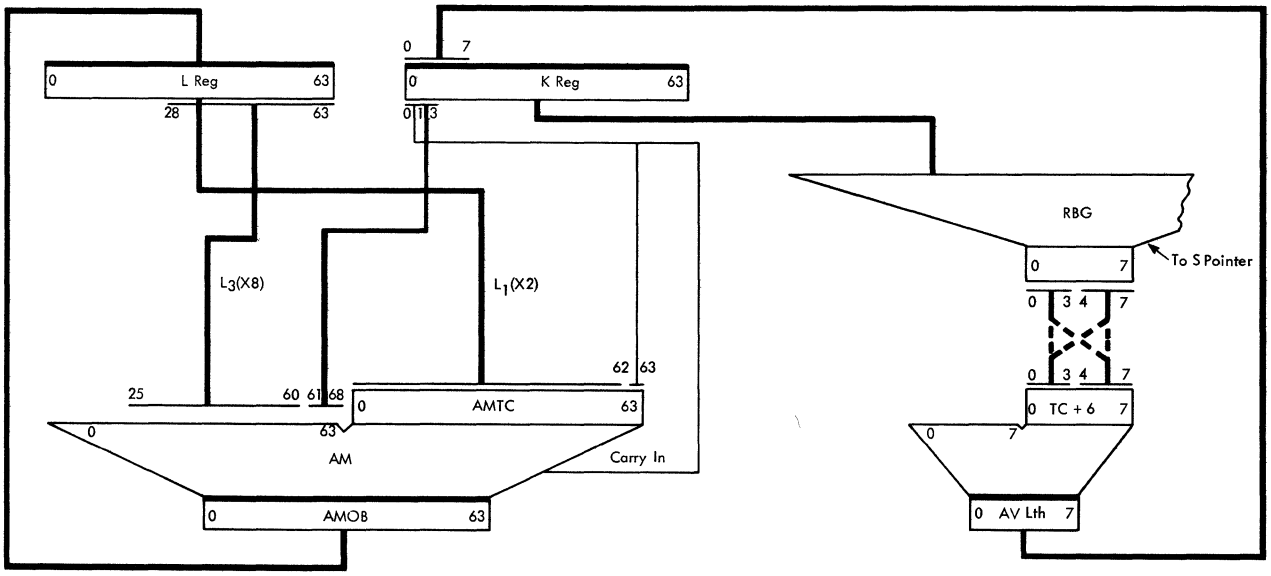
The SC is stepped down 1 at the beginning of each convert cycle. The S pointer steps to 7 at the same time the SC steps to 2. Therefore, the SC steps to 1 at the beginning of the cycle following IS 3; this is the cycle during which the last decimal digit is con-

Decimal Number
1001 1000 0101 = 985

x8 = Shift Left 3
x2 = Shift Left 1

10 X Partial Result (Zero)	x8	0	0	0	0	0	0	0	0	0			
	x2	0	0	0	0	0	0	0	0	0	0	0	
Add First High-Order Digit	(9)									1	0	0	1
Partial Result		0	0	0	0	0	0	0	0	1	0	0	1
10 X Partial Result	x8	0	0	0	0	0	1	0	0	1			
	x2	0	0	0	0	0	0	0	1	0	0	1	
Add Next High-Order Digit	(8)									1	0	0	0
Partial Result		0	0	0	0	0	1	1	0	0	0	1	0
10 X Partial Result	x8	0	0	1	1	0	0	0	1	0			
	x2	0	0	0	0	1	1	0	0	0	1	0	
Add Last Digit	(5)									0	1	0	1
Result		0	0	1	1	1	1	0	1	1	0	0	1

FIGURE 71. CONVERT TO BINARY (EXAMPLE)



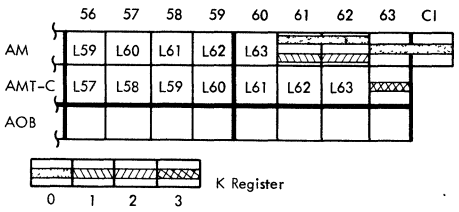


FIGURE 72. CVB DATA GATING

verted. The completed result is on the AMOB during the latter part of this cycle. SC = 1 causes AMOB to be gated into M register, the data gates from K and L registers to AM inputs to be terminated at the end of this cycle.

The last cycle that the convert trigger is on sets the completed result into the M register and steps the SC to 0. During this cycle, the 32-bit completed result is in M register positions 32-63. To prepare the converted result for put-away, it must be transferred to K register positions 0-31. The transfer is made through the main adder with M 32-63 gated L 32 to AMTC 0-31. AMOB 0-63 is then gated to K 0-63. If the sign trigger is on during this cycle, AMTC is gated complement and the completed result enters K register in 2's complement.

PA and ELC: PA 1 cycle follows the convert cycle during which the SC stepped to 0. During the PA 1 cycle, the completed result of the CVB sequence is transferred from K 0-31 to the general register specified by R1. PA and ELC are concurrent; ELC terminates the instructions.

Interrupts

Interrupts that can occur when executing the CVB instruction are:

Address: Occurs if the X2 + B2 + D2 computed storage is invalid. This interrupt occurs prior to the I/E transfer and causes the first E cycle to be ELC to terminate the instruction.

Specification: This interrupt occurs when the computed storage address does not comform to double word boundaries in storage. This interrupt occurs prior to the I/E transfer. ELC is forced during the first E cycle and the instruction is terminated.

Data: A data interrupt occurs during the execution of the CVB instruction whenever an invalid decimal or sign digit is encountered. During IS 2 and IS 3 cycles, each byte gated through the RBG and decimal adder is checked for parity and each digit is checked to be sure it is a valid decimal digit. During the last IS 3 cycle, when S = 7, the LOD from the RBG must be a sign digit.

A data interrupt causes the instruction to be terminated by forcing ELC.

FXP Divide Exception: This interrupt occurs whenever the result exceeds 31 bits. During convert cycles, when the decimal digit is added to 10X the partial result, if a carry-out of position 32 of the AM occurs, a fixed-point divide exception interrupt is signaled.

DIRECT CONTROL (WRD AND RDD) (FIGURE 6496)

The store-fetch sequencers are used for these two instructions. The store-fetch trigger is set with the E go condition, which also sets VFL Seq T2. The sequence is SF 1-5 for both instructions.

The timing for the pulsed signals is generated by OR'ing together three sequencers and their latches. The three sequencers used are A, B and C. To preserve the correct timing when in single-cycle mode, the B and C sequencers are set with the A running clock. This means once the A sequencer is set, the other two follow with the normal timing relationship.

The Y Z counters are set during the last cycle of every instruction, regardless of format, and during every cycle between ELC and the first cycle of the next instruction. Therefore, at the beginning of either RDD, or WRD, Y Z contains IOP (8-15). Y Z are gated to the direct signal out bus with the timing signal described above. This timing signal also generates Read Out and Write Out.

Write Direct (WRD)

Write direct (Figure 6497) fetches a word from storage, puts it in K and gates the addressed byte to the direct control register. The direct control register is set with a running A clock and its release is gated with VFL Seq LA. This maintains the correct relationship between the register setting and rise of the signal out.

Read Direct (RDD)

Read direct (Figure 6498) gates direct in data lines to the input of the AOE. The AOE generates parity, and the AOE output is put in the addressed byte of K. The parity check on K is blocked until the byte read in has been through AOE a second time to generate parity. This prevents a data change at the input to the AOE, just before the latch is locked, from causing a machine check (K register parity error). If the data changed just before the latch locked, the parity generator might not have time to adjust before the fall of the A clock that sets K. As the byte goes from K, to the AOE, to K, the VFL store request trigger is set.

The hold in line being down when SF 1 latch or SF 2 trigger are 1, allows VFL T2 to be set. The VFL T2 latch sets SF 3 and resets SF 2. VFL T2 is used as a buffer to prevent timing malfunctions on the hold in line from causing sequencing faults. VFL T2 latch blocks the release of K so that the byte set in K when VFL T2 was set is the byte stored.

INDEX

FIXED POINT, I EXECUTE, AND BRANCH

FLOATING POINT

VARIABLE FIELD LENGTH (VFL)

COMMENT SHEET

2075 PROCESSING UNIT—VOLUME 3
THEORY OF OPERATION

FIELD ENGINEERING MANUAL OF INSTRUCTION FORM 223-2874-1

FROM

NAME _____ OFFICE/DEPT NO. _____

CITY/STATE _____ DATE _____

To make this manual more useful to you, we want your comments: what additional information should be included in the manual; what description or figure could be clarified; what subject requires more explanation; what presentation is particularly helpful to you; and so forth.

FOLD FOLD

CUT ALONG LINE

FOLD FOLD

How do you rate this manual: Excellent _____ Good _____ Fair _____ Poor _____

Suggestion from IBM Employees giving specific solutions intended for award considerations should be submitted through the IBM Suggestion Plan.

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES, STAPLE, AND MAIL

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y. 12602

ATTN: FE MANUALS, DEPARTMENT B95

FOLD

FOLD

CUT ALONG LINE

STAPLE

STAPLE

IBM **FE Supplement**

System/Unit	2075
Re: Form No.	223-2874-1
This Supplement No.	S26-7035
Date	January 1968
Previous Supplement Nos.	None

This supplement revises and updates Volume 3 of the Field Engineering Manual of Instruction on the IBM 2075 Processing Unit, Form 223-2874-1. This supplement incorporates the floating point changes released under Engineering change 705848E.

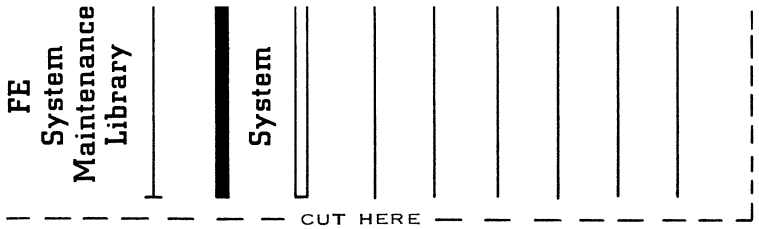
Incorporate this supplement by replacing Title page, Preface page, Contents page, Illustrations page, pages 51 through 56, 59 through 68, 71, 72, 77, and 78, and adding pages 68A and 72A.

Changes to text are indicated by a vertical bar to the left of the affected material. Revised diagrams are identified by a bullet (•) to the left of the figure caption. (In addition, changes that are not readily apparent are indicated by a vertical bar to the left of the changed area.

File this cover letter at the back of the publication. It will then serve as a record of the changes received and incorporated.

International Business Machines Corp., Product Publications Dept., Neighborhood Road, Kingston, N.Y. 12401

PRINTED IN U.S.A.



223-2874-1

Printed in U.S.A. 223-2874-1

IBM

International Business Machines Corporation
Field Engineering Division
112 East Post Road, White Plains, N.Y. 10601