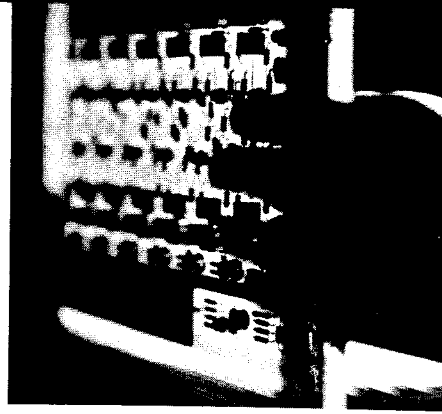
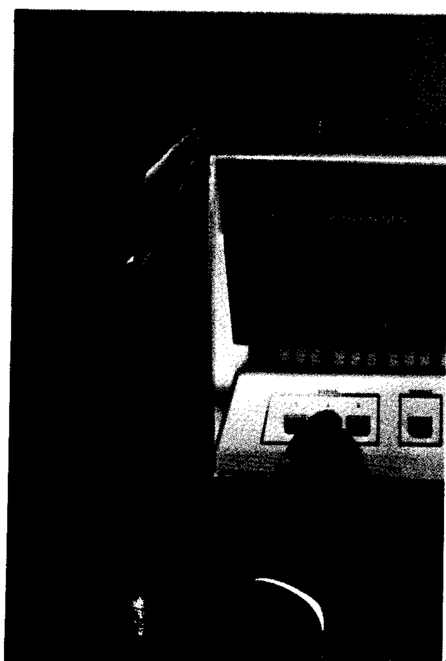
$ 3.00



DATA 620/i

varian data machines
a varian subsidiary

1590 Monrovia Ave., Newport Beach, Calif.
(714) 646-9371 TWX (910) 596-1358



Bulletin #605

Printed in U.S.A. Oct.'67

DATA 620/i systems computer manual

DATA 620/i systems computer manual

varian data machines
a varian subsidiary

DATA 620/i

SYSTEMS COMPUTER

MANUAL

CONTENTS

Page

SYSTEMS COMPUTER

FEATURES OF THE DATA 620/i SERIES COMPUTERS 1

INTRODUCTION 2

The DATA 620/i 2
The DATA 620/i Interface 2
The DATA 620/i User Interface 3

SYSTEM INTERFACE 4

ORGANIZATION 6

Registers 7
Micro-EXEC 8

WORD FORMATS 9

Instruction Word Format 9
Optional Instructions 13

INSTRUCTION LIST 14

MEMORY 18

RELIABILITY AND MAINTAINABILITY. 19

Failure Protection 19
Physical 20
Environmental 20

PROGRAMMED INPUT/OUTPUT 21

Direct Memory Access and Interrupt Logic 21
Interrupt System 22
Buffer Interlace Controller 22
Real-Time Clock 23
Sense Line 23
External Control Lines 23
Parallel I/O Channels 23

Page

SYSTEMS COMPUTER (continued)

PERIPHERAL EQUIPMENT 25

DATA 620/i Series Peripheral Equipment 25

SYSTEM SOFTWARE 27

Symbolic Assembler 27
FORTRAN 27
AID 27
Diagnostic Program Package 27
Subroutine Library 28

USER SERVICES 29

Documentation 29
Programming Training 29
Maintenance Training 29
User Organization 29
Application Programming 30

DATA 620/i SPECIFICATIONS 31

Fully Compatible System Components 33
Micro-VersaLOGIC Integrated Circuit Logic Modules 34
VersaSTORE Core Memories 34
VersaSTORE Mainframe Memories 34

SYSTEM REFERENCE

I INTRODUCTION

1.1 The DATA 620/i 1-1
1.2 Use of the Manual 1-2
1.3 Specifications 1-4

CONTENTS (continued)

Page

SYSTEM REFERENCE (continued)

II DATA 620/i SYSTEM DESCRIPTION

2.1 Computer Organization 2-1
2.2 Computer Word Formats 2-5

III DATA 620/i CENTRAL PROCESSOR INSTRUCTIONS

3.1 General 3-1
3.2 Single-Word Instructions 3-1
3.3 Double-Word Instructions 3-30

IV DATA 620/i INPUT/OUTPUT SYSTEM

4.1 Introduction 4-1
4.2 Organization 4-1
4.3 Program Control Functions 4-6
4.4 Automatic Control Functions 4-12

V CONTROL CONSOLE OPERATION

5.1 Controls and Indicators 5-1
5.2 Manual Operations 5-4

PROGRAMMING REFERENCE

I GENERAL DESCRIPTION

1.1 Introduction 1-1
1.2 Purpose of the Manual 1-2
1.3 Computer Operation 1-2

II DATA 620/i ASSEMBLY SYSTEM

2.1 Introduction 2-1
2.2 DAS Source Language 2-1
2.3 DATA 620/i Instructions 2-7

Page

PROGRAMMING REFERENCE (continued)

2.4 DAS Pseudo Instructions 2-15
2.5 Source Statement Formats 2-28
2.6 DAS Output List 2-29
2.7 Operating the DAS Assembly System 2-32
2.8 FORTRAN Pseudo Instructions 2-32

III AID-UTILITY AND DEBUGGING PACKAGE

3.1 Introduction 3-1
3.2 Bootstrap Loader 3-1
3.3 Binary Load/Dump 3-3
3.4 AID II Package for the DATA 620/i 3-4

IV SOURCE TAPE CORRECTION PROGRAM

4.1 Introduction 4-1
4.2 Operating Procedures for COR 4-1

FORTRAN REFERENCE

I BASIC FORTRAN CONCEPTS

1.1 Introduction 1-1
1.2 Character Set 1-1
1.3 Line Format 1-2

II DATA

2.1 General 2-1
2.2 Data Types 2-1
2.3 Data Names 2-1
2.4 Variables 2-2
2.5 Constants 2-2
2.6 Arrays 2-3

CONTENTS (continued)

Page

FORTRAN REFERENCE (continued)

III SPECIFICATION STATEMENTS

3.1 General 3-1
3.2 DIMENSION Statement 3-1
3.3 COMMON Statement 3-2
3.4 EQUIVALENCE Statement 3-3

IV EXPRESSIONS AND STATEMENTS

4.1 Arithmetic Expressions 4-1
4.2 Arithmetic Assignments and Replacements 4-3

V CONTROL STATEMENTS

5.1 General 5-1
5.2 GO TO Statements 5-1
5.3 Arithmetic IF Statement 5-2
5.4 CALL Statement 5-3
5.5 RETURN Statement 5-3
5.6 CONTINUE Statement 5-4
5.7 PAUSE Statement 5-4
5.8 STOP Statement 5-4
5.9 DO Statement 5-5

VI INPUT/OUTPUT STATEMENTS

6.1 General 6-1
6.2 Input/Output Lists 6-1
6.3 Simple Lists 6-1
6.4 DO - Implied Lists 6-2
6.5 READ Statements 6-2
6.6 WRITE Statements 6-3
6.7 REWIND Statements 6-4
6.8 BACKSPACE Statements 6-4
6.9 ENDFILE Statements 6-4
6.10 FORMAT Statements 6-4
6.11 Field Specifications 6-5

Page

FORTRAN REFERENCE (continued)

6.12 F Conversion 6-5
6.13 E Conversion 6-6
6.14 I Conversion 6-7
6.15 H Conversion 6-8
6.16 X Specification 6-9
6.17 / Specification 6-10
6.18 REPEAT Specification 6-10
6.19 Format Control and List Interaction 6-11

VII PROGRAMS AND SUBPROGRAMS

7.1 General 7-1
7.2 Main Programs 7-1
7.3 Subprograms 7-1
7.4 Statement Functions 7-2
7.5 Intrinsic Functions 7-2
7.6 Function Subprograms 7-4
7.7 Basic External Programs 7-5
7.8 Subroutine Subprograms 7-5
7.9 Dummy Arguments 7-7

VIII FORTRAN OPERATING INSTRUCTIONS

8.1 General 8-1
8.2 Compiler Operating Instructions 8-1
8.3 Preliminary Operations 8-1
8.4 Normal Operations 8-1
8.5 Input Records 8-2
8.6 Output Records 8-2
8.7 Notification Errors 8-3
8.8 Terminating Errors 8-3
8.9 Optional Listings 8-4
8.10 Program Map 8-4
8.11 FORTRAN Loader Operating Instructions 8-4
8.12 Preliminary Operations 8-4
8.13 Loading Subprograms 8-5
8.14 Error Diagnostics 8-5
8.15 Execution of FORTRAN Programs 8-6

Page

FORTRAN REFERENCE (continued)

8.16 Programmed Halts 8-6
8.17 Error Bit Designators 8-6
8.18 Error Halts 8-7
8.19 Binary Input/Output 8-7
8.20 BCD Input/Output 8-7

IX GLOSSARY

SUBROUTINE DESCRIPTIONS

I GENERAL DESCRIPTION

1.1 Introduction 1-1
1.2 Programming Standards 1-1

II PROGRAM DESCRIPTION

2.1 Introduction 2-1
2.2 Identification 2-1

III PROGRAMMED ARITHMETIC

IV ELEMENTARY FUNCTIONS

V UTILITY AND DEBUGGING ROUTINES

VI EXECUTIVE ROUTINE

INTERFACE REFERENCE

I GENERAL DESCRIPTION

1.1 Introduction 1-1
1.2 Purpose of the Manual 1-2
1.3 Computer Organization 1-3

CONTENTS (continued)

Page

INTERFACE REFERENCE (continued)

II DATA 620/i STANDARD INPUT/OUTPUT SYSTEM

2.1 Organization 2-1
2.2 Program Control Functions 2-12
2.3 Automatic Controlled Functions 2-24
2.4 Miscellaneous Signals 2-30

APPENDICES

A DATA 620/i NUMBER SYSTEM

B STANDARD DATA 620/i SUBROUTINES

C TABLE OF POWERS OF TWO

D OCTAL-DECIMAL INTEGER CONVERSION TABLE

E OCTAL-DECIMAL FRACTION CONVERSION TABLE

F DATA 620/i INSTRUCTIONS (ALPHABETICAL ORDER)

G DATA 620/i INSTRUCTIONS (BY TYPE)

H DATA 620/i RESERVED INSTRUCTION CODES

I STANDARD CHARACTER CODES

J TELETYPE I/O INSTRUCTIONS

K FORTRAN STATEMENT TYPES

L FORTRAN I/O UNIT ASSIGNMENTS

M FORTRAN MEMORY MAPS

N FORTRAN OBJECT RECORD FORMAT

SYSTEMS COMPUTER

FEATURES OF THE DATA 620/i SERIES COMPUTERS

Field Proven Software

Silicon Monolithic Integrated Circuits (DTL and TTL)

9 Hardware Registers

Over 100 Basic Commands

6 Addressing Modes

Direct Addressing to 2,048 or 32,768 Words

16- or 18-Bit Words

Expansion to 32,768 Words

Hardware Index Registers

Party Line I/O Facility

Micro-EXEC Option

10-1/2 Inches of Rack Space

Less than 70 Pounds (Mainframe and power supply)

340 Watts

NPN or PNP (Optional) I/O Levels

Interface Ease

Compatible with DATA 620 Computer

Plug-In Expandable

Low Cost

INTRODUCTION

DATA 620/i is a system-oriented digital computer, designed as a powerful system computer to fill the gap between special purpose digital hardware and general purpose computers. DATA 620/i meets all the requirements of a true system computer – powerful computing ability, easy interfacing, modular design and construction for expandability, integrated circuit reliability, low cost, and compact size.

In addition, DATA 620/i offers a number of features simply not available on other computers – like party line communication, quick and easy memory expandability from 4,096 to 32,768 words of 16 or 18 bits, and a unique micro-EXEC microstep sequencing technique. DATA 620/i comes with a complete set of field-proven software, developed and perfected on the DATA 620.

DATA 620/i has a bigger instruction set, 1/2 the components, and costs less than any computer in its class. This is why it so efficiently and economically solves system problems previously considered too difficult or expensive for computer solution.

DATA 620/i offers a wide variety of peripherals and options, allowing the user to select only those features specifically required for his application, and providing the optimum amount of computer power per dollar.

THE DATA 620/i

As a physical system component, DATA 620/i processors are compact in size, occupying only 10.5 inches of rack space. They are accessible from the front like other system components, and they are reliable and maintainable. The contents of five operational registers can be displayed on the front panel.

Eighty-five percent of the processor operation can be verified from the front panel without the use of an oscilloscope. As the controlling element in a system, a DATA 620/i has the "raw" data manipulating power of a much more costly computer. The instruction set includes over 100 basic machine commands. The register change command is micro-programmable with over 100 useful combinations. The processing characteristic can be adapted to specific requirements through an optional Micro-EXEC facility that permits software programs to be hardware implemented.

THE DATA 620/i INTERFACE

The DATA 620/i series was designed to not only provide the complete spectrum of interface capabilities required in a system computer, but to also allow the user to tailor the computer for his specific application. To attain this goal, all of the Input/Output features are offered as options. Among these facilities are: direct memory access, real-time clock, power failure protect, and the buffer interlace controller.

These features, combined with priority interrupts, external sense lines, external control lines and the proprietary Micro-EXEC technique give the DATA 620/i family virtually every I/O capability available.

THE DATA 620/i USER INTERFACE

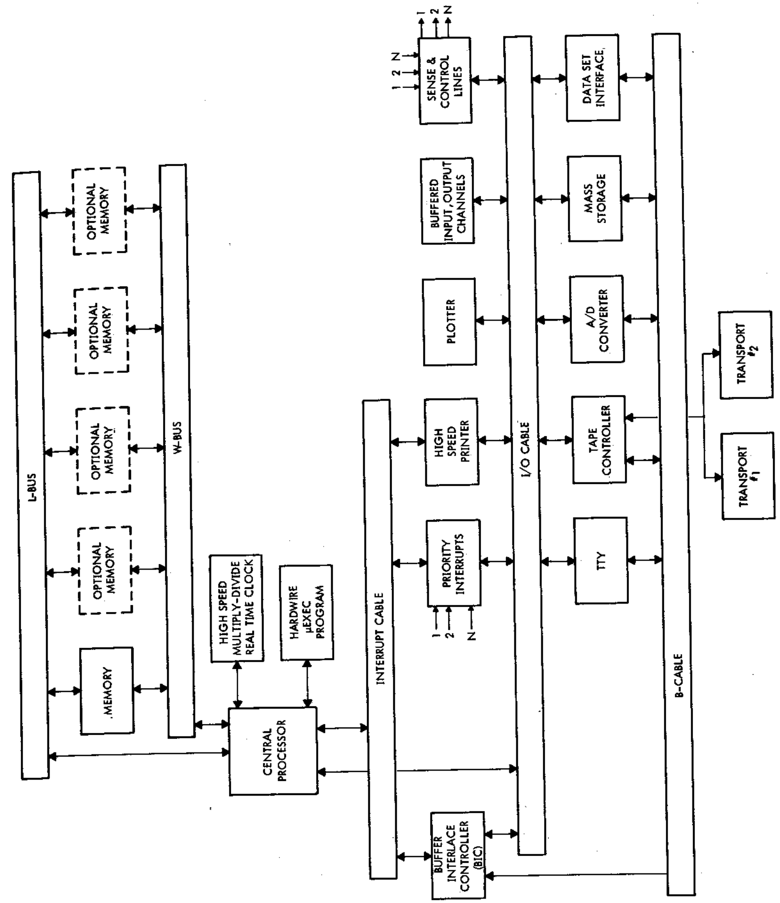
As must be the case in any machine that is required to do – and do well – a large number of data manipulation tasks which are unspecifiable in advance, flexibility was the motif in designing the DATA 620/i software package. The goal was to achieve flexibility without creating big problems on the one hand, or falling into the easy habit of accepting hardware/software tradeoffs on the other hand. In the DATA 620/i, hardware and software features reinforce each other. For example, there are five modes of single-word addressing, one of which permits direct addressing of four times as many words in store as is normally possible with conventional designs. Multiply/divide instructions are available as options to meet more demanding computation speed requirements.

SYSTEM INTERFACE

The ability of the computer to adapt to the system is an excellent criterion for determining a true systems computer.

The design philosophy behind the 620/i input/output structure is not only to provide all of the capabilities needed in a system computer, but to allow the user to choose the particular capability needed for his particular application. The reasoning is: if the feature is needed, it can be provided as a low cost option; if the need is uncertain, it can be easily added in the field if and when it is needed.

The DATA 620/i family offers the widest range of interface facilities. These include party line communication bus, multilevel priority interrupts, external sense lines, external control lines, direct memory access, and interlace control.



DATA 620/i Organization

ORGANIZATION

The DATA 620/i is organized with a unique bus structure, selection logic, and nine registers. The organization provides universal internal information routing, buffered processing, micro-register change programming facility, information indexing without time penalty, and the optional direct memory access (cycle stealing) facility..

The organization optimizes the DATA 620/i for maximum I/O throughout, minimum elapsed time between successive input or output transfers, and minimum programming.

This unique organization makes possible the optional Micro-EXEC facility by which complex algorithms or additional instructions can be implemented with external hardware. The Micro-EXEC technique produces an increase in processing speed in excess of 500 percent over conventional stored program techniques. The bus structure of this computer family permits the system designer to overcome traditional barriers of processing speed, high-rate volume throughput, and fixed mainframe characteristics. The four available busses are:

L bus provides a 12-bit parallel communications path from the L register to the address decoders in the memory modules.

W bus provides a parallel data communications path (16/18 bit) from the W register to the memory module(s) (up to 8).

C bus provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O unit, and the operational registers. This bus permits data to be uniquely or commonly transferred to the operational registers. It performs the distribution function for micro-programming, and provides a bi-directional parallel word path to the "party line".

C bus is the central communication avenue and connects with all internal units of the processor. It is the key facility that permits Micro-EXEC to be implemented.

S bus provides the parallel path and selection logic for routing data between the operational registers and the arithmetic unit. It implements the select, gather, and route function for micro-programming and Micro-EXEC.

Party line I/O bus provides a 16/18-bit parallel bi-directional I/O communication path. This bus includes the control lines for transfer ready, sense, control, interrupt address and acknowledge, and information entry. The "party line" is packaged as one cable, and each peripheral device has a party line connector and a party line extender connector. The device and the party line form a "daisy chain" whereby additional I/O controllers can be added on site and on a plug-in basis.

REGISTERS

Nine registers are provided with a basic processor. Four of the nine registers are incorporated to provide buffering to satisfy real-time system requirements. All the arithmetic and control unit registers are multipurpose and can serve a unique micro-programming and Micro-EXEC function.

A register is a full-word register and is the high-order half of the accumulator. A is a source and destination for programmed input/output and micro-programming. Micro-EXEC can select, set, shift, and perform arithmetic and logical operations on A.

B register is a full-word register and is the low-order-half of the accumulator. B is a source and destination for programmed input/output, is micro-programmable, and can serve as the second hardware index register. Micro-EXEC can select, set, shift, and perform arithmetic and logical operations on the A.

X register is a full-word register which permits indexing of memory addressing without adding time to accessing an indexed location. The X register is addressable by the micro-programming instruction set where it serves logical, storage and counting functions. Micro-EXEC can use the X register for arithmetic and multiple other functions.

P register is a full-word register and is the program counter. P can serve multiple purposes under Micro-EXEC.

U register is a full-word buffer which holds the instruction being executed. The U register buffers the control unit from memory to permit interlace I/O operation to occur on a memory-cycle by memory-cycle-basis. It is also a multipurpose register available to Micro-EXEC.

S register is a 5-bit register which, in combination with the U register controls the length of shift instructions. This register also buffers memory from the control unit. S register is available to Micro-EXEC.

L register is the 12-bit memory location register. Micro-EXEC can select and set the L register.

W register is the memory word register and is full length (16 or 18 bits). W is selectable and can be set by Micro-EXEC.

R register is a full-word buffer which holds the multiplicand and divisor, in arithmetic operations. R register buffers the arithmetic unit from memory to permit interlace I/O operations to occur on a memory-cycle-steal basis. It is also a multipurpose register available to Micro-EXEC.

Micro-EXEC

Micro-EXEC (optional) is a technique by which the system designer has the option of externally combining and sequencing the processor's micro-steps to perform a complex macro-function. Over 30 micro-step control lines are made available to the system user. These control functions are the micro-steps normally controlled by machine instructions.

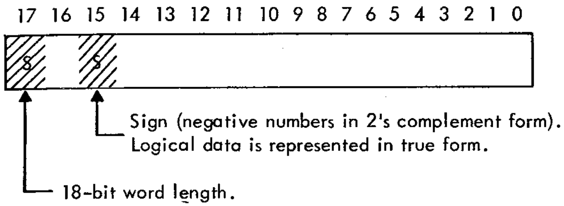
They control memory, arithmetic unit, control unit, all registers, I/O and communication networks. The external control can operate the micro-steps as fast as five every 900 nanoseconds by utilizing the processor clock to synchronize the micro-step operations. Micro-EXEC can be used to implement many types of algorithms. Typical functions are: convolutions, coordinate transformations, double precision arithmetic, table look ups, square root, limit checking, etc. Micro-control can produce up to 10-to-1 speed advantage over stored programs and does not require core memory for the program. Opening new dimensions to the data system designer, Micro-EXEC makes practical an extremely fast processor with small or large memories. It permits the mode of processing to be controlled externally, and processing to be optimized for the system.

The processor organization and hardware provides the system engineer with the most flexibility available in off-the-shelf equipment. The standard options of Micro-EXEC, machine instructions, memory, and I/O facilities provide functional adaptability and system optimization without engineering risk or unpredictable costs.

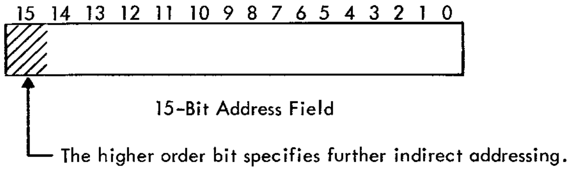
WORD FORMAT

The word formats separate into two categories: data and instruction. Each category has been optimized for the system environment. DATA 620/i processors are available in 16- or 18-bit word length. The 16-bit is the DATA 620/i; the 18-bit version is the DATA 620/i. The data format is extendable for 18-bit words with the sign bit in the high-order positions.

DATA WORD FORMAT



INDIRECT ADDRESS FORMAT

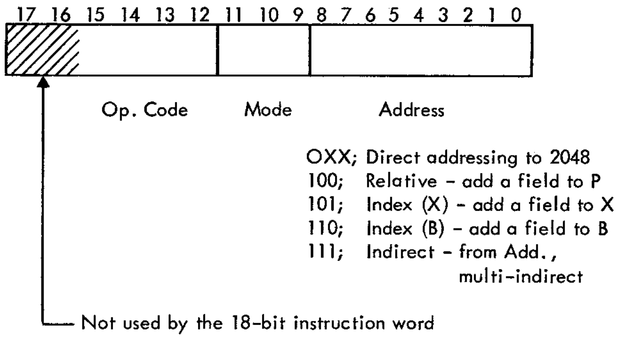


INSTRUCTION WORD FORMAT

The four instruction word formats – single word, double word, generic and macro-command – are illustrated in the following paragraphs.

1. Single word. Twelve basic commands and two optional commands have single word memory reference formats. The single word instruction is divided into three fields as shown below. There are six addressing modes including direct addressing to 2,048 words, relative to P with a delta range of 512, index by X or B, indirect from the contents of the memory location addressed, immediate.

SINGLE WORD INSTRUCTION FORMAT



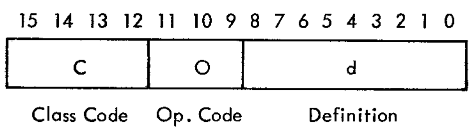
Single Word Instructions include: LDA LDB LDX INR ADD SUB MUL* STA STB STX ERA ORA ANA DIV*.

All basic single word instructions are executed in two cycles, including relative and index addressing modes. One cycle is added for each level of indirect addressing.

The single word instruction format is designed to enable the system user to write his programs in the minimum number of memory locations and have his program executed in minimum time. The format is uncomplicated and the fields divide into convenient octal groupings so that programs can be written and checked rapidly.

2. Generic. Twenty-six instructions are single word generics and divide into the three fields of class code, operation code and definition.

GENERIC INSTRUCTION FORMAT



These instructions perform arithmetic unit, control unit and input/output functions. The operations are: HLT, NOP, shifts (12), overflow (2), sense, external functions, input and output, A or B (11).

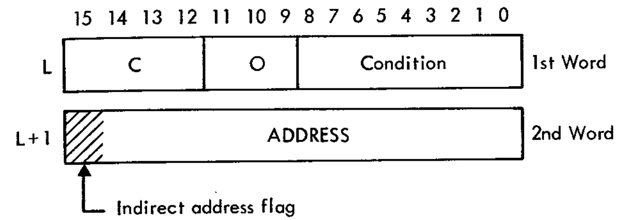
*Optional instruction

The shift instructions can shift up to 32 places. The sense and external function instructions can address up to 64 peripheral devices and define up to 8 functions. The input and output commands can select A or B, A and B; clear and input to A or B, A and B. The input/output instructions can address up to 64 devices. (The in-memory and out-memory instructions and the interrupt priority control are two word instructions.)

The generics are octal grouped for user convenience. They provide flexibility to optimize input/output processing.

3. Two word. Two classes and six types of instructions are two word instructions. The types include: jump, jump and mark, execute, immediate, in/out memory, sense.

JUMP, JUMP and MARK, EXECUTE



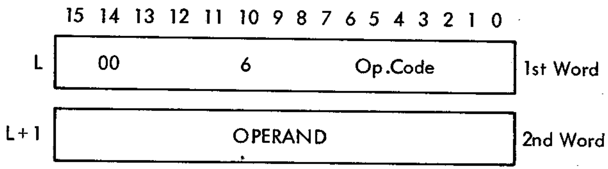
The first word contains three fields: The C field contains the class code, the O field contains the operation code, and the condition field specifies any combination of nine conditions. The nine conditions are: SS1, SS2, SS3, XO, B O, A O, A neg., A pos., and overflow. The second word contains the jump address, jump mark address, or the address of the instruction to be executed. Indirect addressing is permitted. If the specified conditions are all met, the instruction is executed. If the conditions are not met, the second word is skipped and the P register incremented.

The in/out memory has a similar two word instruction format. The condition field of the INM/OTM instruction addresses the device selected; the second word contains the memory address for the data. Indirect addressing is permitted.

Immediate is a special type instruction. The type includes twelve (plus two optional) two word instructions. The instructions include: LDAI LDBI LDXI ADDI SUBI INRI MULI* STAI STBI STXI ERAI ORAI AWAI DIVI*.

*Optional

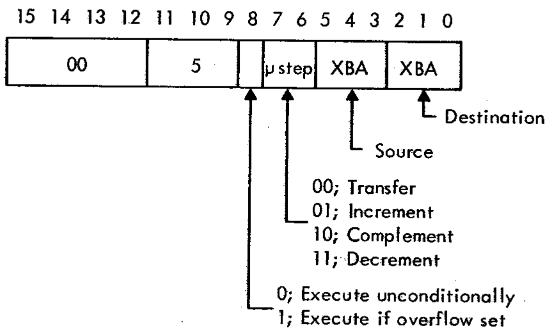
IMMEDIATE INSTRUCTION FORMAT



4. Macro-commands. A number of micro-steps are programmable into a macro-instruction with the single word "macro-command." This command has over 128 useful combinations including those listed in the instruction set. The macro-command format is:

> Bits 3 through 6 define one of the instructions above. The immediate type instructions provide literal addressing. Literal addressing, being the operand address field, contains the operand. This type automatically increments the P counter; after the execution, the next instruction is obtained from P + 2.

There are a total of 45 standard instructions and over 16 optional two word instructions. The efficiency and power of the two word instructions becomes more and more apparent with use. They provide direct and random addressing and accessing to 32,768 words. In most cases, they permit a two memory location sequence of instruction to replace the usual three memory location sequence. The amount of memory conserved and time saved by these instructions depends on the application, and ranges from 5 to 25 percent.



The X, B, and A register contents can be logical "ORed," cleared, transferred, set to a common value, complemented, "NORed," incremented, decremented, and, if desired, conditionally on an overflow. Sequences of micro-commands can be used to perform additional logical functions customary in a system environment.

OPTIONAL INSTRUCTIONS

The hardware multiply/divide and extended addressing option provides an additional 16 instructions to the basic instruction set. The extended address mode is similar in format to the immediate address instructions, except that the second word of the double-word instruction contains the effective address. All single word commands can use extended addressing.

The instruction set is the most comprehensive available with "compact" computers or processors. The optional instruction sets have specific value to certain applications and are available to refine the processors to those applications. The instruction set, variety, simplicity, and power equates to economic optimization. The instruction list is presented in the following table.

INSTRUCTION LIST

TYPE	MNEMONIC	DESCRIPTION	TIME CYCLES
Load	LDA	Load A Register	2
	LDB	Load B Register	2
	LDX	Load X Register	2
Store	STA	Store A Register	2
	STB	Store B Register	2
	STX	Store X Register	2
Arithmetic	ADD	Add to A Register	2
	SUB	Subtract from A Register	2
	INR	Increment and Replace	3
	MUL*	Multiply B Register, Double Length	10
	DIV*	Divide AB Register, Double Length	10-14
Logical	ERA	Exclusive OR to A Register	2
	ORA	Inclusive OR to A Register	2
	ANA	And to A Register	2
Jump	JMP	Jump Unconditionally	2
	JOF	Jump if Overflow Set	2
	JAN	Jump if Register Negative	2
	JAZ	Jump if A Register Zero	2
	JAP	Jump if Register Positive	2
	JSS1	Jump if Sense Switch 1 is Set	2
	JSS2	Jump if Sense Switch 2 is Set	2
	JSS3	Jump if Sense Switch 3 is Set	2
	JXZ	Jump X Register Zero	2
	JBZ	Jump B Register Zero	2
Jump and Mark	JMPM	Jump Unconditionally and Mark	2
	JOFM	Jump Overflow Set and Mark	2-3
	JANM	Jump A Register Negative and Mark	2-3
	JAZM	Jump A Register Zero and Mark	2-3
	JAPM	Jump A Register Positive and Mark	2-3
	JAS1M	Jump Sense Switch 1 Set and Mark	2-3

INSTRUCTION LIST (continued)

TYPE	MNEMONIC	DESCRIPTION	TIME CYCLES
Jump and Mark (continued)	JS2M	Jump Sense Switch 2 Set and Mark	2-3
	JS3M	Jump Sense Switch 3 Set and Mark	2-3
	JXZM	Jump X Register Zero and Mark	2-3
	JBZM	Jump B Register Zero and Mark	2-3
Execute	XEC	Unconditional Execute	2
	XOF	Execute Overflow Set	2
	XAN	Execute A Register Negative	2
	XAZ	Execute A Register Zero	2
	XAP	Execute A Register Positive	2
	XS1	Execute Sense Switch 1 Set	2
	XS2	Execute Sense Switch 2 Set	2
	XS3	Execute Sense Switch 3 Set	2
	XXZ	Execute X Register Zero	2
	XBZ	Execute B Register Zero	2
Immediate	LDAI	Load A Register Immediate	2
	LDBI	Load B Register Immediate	2
	LDXI	Load X Register Immediate	2
	STAI	Store A Register Immediate	2
	STBI	Store B Register Immediate	2
	STXI	Store X Register Immediate	2
	ADDI	Add to A Register Immediate	2
	SUBI	Subtract from A Register Immediate	2
	MULI*	Multiply B Register Immediate Double Length	10
	DIVI*	Divide AB Register Immediate Double Length	10-14
	INRI	Increment and Replace Immediate	3
	ERAI	Exclusive OR to A Register Immediate	2
	ORAI	Inclusive OR to A Register Immediate	2
	ANAI	And to A Register Immediate	2
Input/Output	EXC	External Control Function	1
	CIA	Clear and Input to A Register	2

TYPE	MNEMONIC	DESCRIPTION	TIME CYCLES
Input/Output (continued)	CIB	Clear and Input to B Register	2
	CIAB	Clear and Input to A and B Registers	2
	INA	Input to A Register	2
	INB	Input to B Register	2
	INAB	Input to A and B Registers	2
	IME	Input to Memory	3
	OAR	Output A Register	2
	OBR	Output B Register	2
	OAB	Output OR or A and B Registers	2
	OME	Output from Memory	3
	SEN	Sense Input/Output Lines	2.25
Register Change	IAR	Increment A Register	1
	DAR	Decrement A Register	1
	IBR	Increment B Register	1
	DBR	Decrement B Register	1
	IXR	Increment X Register	1
	DXR	Decrement X Register	1
	CPA	Complement A Register	1
	CPB	Complement B Register	1
	CPX	Complement X Register	1
	TAB	Transfer AR to B Register	1
	TBA	Transfer BR to A Register	1
	TAX	Transfer AR to X Register	1
	TBX	Transfer BR to X Register	1
	TXA	Transfer XR to A Register	1
	TXB	Transfer XR to B Register	1
	TZA	Transfer Zero to A Register	1
	TZB	Transfer Zero to B Register	1
	TZX	Transfer Zero to X Register	1
	AOFA	Add OF to A Register	1
	AOFB	Add OF to B Register	1
	AOFX	Add OF to X Register	1
	SOFA	Subtract OF from A Register	1
	SOFB	Subtract OF from B Register	1
	SOFX	Subtract OF from X Register	1
	SOF	Set Overflow	1
	ROF	Reset Overflow	1

TYPE	MNEMONIC	DESCRIPTION	TIME CYCLES
Logical Shift	LSRA	Logical Shift Right A k places	1 + 0.25k
	LRLA	Logical Rotate Left A k places	1 + 0.25k
	LSRB	Logical Shift Right B k places	1 + 0.25k
	LRLB	Logical Rotate Left B k places	1 + 0.25k
	LLSR	Long Logical Shift Right k places	
	LLRL	Long Logical Rotate Left k places	1 + 0.25k
Arithmetic Shift	ASRA	Arithmetic Shift Right A k places	1 + 0.25k
	ASRB	Arithmetic Shift Right B k places	1 + 0.25k
	ASLA	Arithmetic Shift Left A k places	1 + 0.25k
	ASLB	Arithmetic Shift Left B k places	1 + 0.25k
	LASR	Long Arithmetic Shift Right k places	1 + 0.25k
	LASL	Long Arithmetic Shift Left k places	1 + 0.25k
CONTROL	HLT	Halt	1
	NOP	No Operation	1

*Denotes optional instruction. Times given are for 16-bit computer.
Add 1 cycle for each level of indirect addressing.

MEMORY

The DATA 620/i uses general purpose random access ferrite magnetic core memories. They contain a proprietary thermal compensation technique which preserves the operating margins over the temperature range (0° to 45°C) without adjustment.

The memory communicates with the processor through a memory data bus and an address bus. Additional external (to mainframe) memory modules can be added simply by adding an optional memory adapter to the processor that permits the additional module to be "plugged in." The external memory module includes an adapter for the next memory module. The memory can be expanded to 32,768 words by the addition of 4K memory modules.

Memory cycle time is 1.8 microseconds; access time is 700 nanoseconds.

RELIABILITY AND MAINTAINABILITY

DTL and TTL integrated circuits are used throughout the DATA 620/i. These integrated circuits are general purpose digital logic, and are noted for low power consumption, high packing density, high noise rejection, and reliability throughout the operating temperature range of 0° to 45°C. The low power equates to low heat generation and high reliability.

DATA 620/i computers are produced under a quality control program designed and practiced to meet MIL-Q-9858A, and to the intent of NPC 200-3. The mean-time-between-failures (MTBF) has been calculated for the basic processors to be over 7,500 hours. The mean-time-to-repair is estimated to be a few minutes.

DATA 620/i computers are packaged to simplify maintenance. The integrated circuit board layout is unique using a "bit slice" alyout. Bit slice is a technique whereby all register and gating circuits associated with six bits are packaged on one card.

The structure is designed for easy access. All units of the processor are mounted to be easily removed to make all components and wiring easily accessible. The "big board" concept is used to permit easy trouble shooting.

FAILURE DETECTION

The source of faults in solid-state electronic equipment with conservative circuit and timing designs is from external causes. The external causes are power failures, power frequency failures, excessive heat and the failure of electro-mechanical peripheral devices. The DATA 620/i has been designed to prevent each of these fault sources from destroying the integrity of the system computer function.

1. Power failure. An optional power failure protect system monitors power line voltage. If voltage is outside safe limits, a power fail interrupt is generated. The interrupt subroutine assures an orderly, safe shutdown. Upon restoration of power, the computer is automatically restarted at a designated memory location, and appropriate software provides an orderly restart.

2. Temperature. A thermal sensor is embedded in the core memory to continually monitor internal temperature. If the temperature rises above the specified limit (45°C), the sensor produces a thermal alarm signal that is used to light the console alarm indicator and/or generate an interrupt line.

3. Operator errors. The control panel is electrically disconnected during run mode.

4. Memory protect. This option permits a top-priority executive, control, alarm, processing, or monitor system to remain resident in memory while other programs are being processed.

These facilities provide the system engineer with the level of assurance needed to tackle the most demanding process control or real-time application where one failure can be extremely costly.

PHYSICAL

1. Packaging. The DATA 620/i family is packaged to offer the user maximum convenience, positioning, flexibility and space-saving economies. The memory, arithmetic and control unit, and the power supply and control console are three separate packages that, when connected, produce a compact unit that is 10-1/2 inches high 22 inches deep and 19 inches wide. The compactness and light weight of the DATA 620/i series enables it to be used in facilities such as submarines, aircraft, etc.

2. Control panel. The user-oriented design philosophy of the DATA 620/i console utilizes sound human engineering practices. The console has been developed to produce a pleasing image and still be functionally easy to use. Proximity of related functions, minimum reflectivity, and other more subtle features such as length and distance of switches were used in the development of the console. The basic function of the console – to modify and monitor all operational registers – was achieved without a cluttering of switches that tend to confuse. A simple straightforward instrument is the result.

ENVIRONMENTAL

The DATA 620/i connects to standard commercial single-phase 115-vac power. Power regulation is not required under normal commercial power conditions. Subflooring or conditioned air are not required. The DATA 620/i is equally at home in the shop, field, instrumentation room, classroom, and laboratory.

PROGRAMMED INPUT / OUTPUT

The basic DATA 620/i processor is equipped with positive voltage level party line I/O bus. The party line is a bidirectional common communication channel containing the data and control lines required for system communication. Time-shared between the peripherals, it is designed to prevent conflicts or traffic jams under heavy communication loads. Each transmission contains the routing information as well as the data. It is transmitted as an entity which is not separable by interrupt. Thus, numerous devices can time share the party line. The transmission has two phases: The first phase is the route set-up, the second is the data transmission.

The party line permits plug-in expansion of all peripheral devices. The party line contains line drivers and line receivers to service up to ten peripheral devices. Each peripheral device contains a data buffer and party line adapter. Thus, no device can tie-up the party line, and modifications to the computer are not required to add peripherals. Each device has a party line connector and a party line extender connector. The last device on the party line has a termination shoe on the extender connector. When another device is added, a party line cable is provided between the added and the last device. The termination shoe is moved to the added device.

The party line technique solves the troublesome problems usually encountered in time-shared operation and on-site system expansion.

The following types of I/O commands can be executed with the basic machine:

Sine Word to/from Memory
Single Word to/from A and B Registers
Test External Sense Line
Generated External Control Line

The following interface features can be added to the basic party line.

DIRECT MEMORY ACCESS AND INTERRUPT LOGIC

This option provides direct memory access (cycle steal capability) from the party line I/O bus. With this feature, the user can design special system devices that cause the program to hesitate for 2.7 microseconds, during which time memory is accessed for data, or data is stored in memory. This trap operation bypasses the A, B, X and P registers, thus allowing the program to proceed normally. One interrupt level is provided with the option.

INTERRUPT SYSTEM

The DATA 620/i has a multilevel priority interrupt system with single-instruction execute, group enable/disable, and selective arm/disarm capability. Each interrupt line is assigned a unique memory destination address that is the first of a pair of locations. The system is modular and expandable in groups of eight or sixteen levels up to 64 levels.

The interrupt system is automatically scanned every 1.8 microseconds and the interrupt is recognized before the fetch cycle of the next instruction to be executed. If signals exist on one or more interrupt lines, the highest priority is recognized. An interrupt functional response to an external device can be accomplished in as little as two memory cycles.

BUFFER INTERLACE CONTROLLER

Many system devices require computer facilities to transmit I/O data at high rates and volumes and at random periods. Such devices are best serviced with automatic channels which do not require programming or interfere with the processing. The buffer interlace controller (BIC) unit option services such requirements.

The BIC contains two 15-bit registers, the party line addressing and control logic, priority logic, and DATA 620/i control logic. The two registers contain the stop address and the current memory address. These registers are set by the program with the start address and the stop address. These addresses define the sequential locations in memory from or to which the data is communicated. Connecting the desired controller to the BIC activates the BIC. The I/O operation is automatic thereafter until the stop address has been met. Each data word transferred requires less than two memory cycles. Information can be transferred at a rate over 200,000 words per second. The BIC automatically synchronizes the data transmission rate to the device requirement.

The BIC connects to the party line and controls the data transmission of the devices with BIC adapters when operating in the interlace mode. Interlace I/O occurs on a memory cycle basis and shares priority with the control processor. The BIC will capture the next memory cycle and stall the computer for 2.7 microseconds for each word transmitted. The processing resumes automatically at the completion of the word transferred. Any device connected to the BIC can be operated under control of the BIC or under program control. Up to eight devices can be connected to one buffer interlace controller unit. The current address can be read under program control.

Each group of eight or sixteen interrupts can be enabled/disabled, and contains a 16-bit mask register that controls the individual interrupt lines. The program can maintain the hardware order of priority or reorder to meet dynamic queuing.

REAL-TIME CLOCK

The DATA 620/i real-time clock is an option that provides a flexible time-orientation system that can be used in a variety of real-time functions, including time-of-day accumulation and as an interval timer.

The real-time clock consists of two interrupts. The first interrupt is a time-base signal that when recognized by the computer, executes an increment memory instruction stored in the interrupt address. The second interrupt occurs when the incremented memory location reaches a count of $40,001_8$.

Acknowledgement of an interrupt by the central processor causes the instruction located memory destination address of the interrupt to be executed. The instruction can be any of the DATA 620/i instruction set. This technique permits the interrupts to be of the single-execute type, whereby single-instruction responses to external signals can be serviced in one instruction period. If the executed instruction is a jump and mark (JMPM), the interrupt system is automatically inhibited to permit the inhibit to be terminated under program control. The DATA 620/i interrupt system provides the high speed reaction time, expansion capability, priority and queuing versatility required for real-time control.

SENSE LINE

Discrete sense lines are available as options in sets of eight. Each sense line has a unique address. Up to 512 sense lines can be addressed. The sense instruction is a two word conditional jump command. If a signal exists on the sense line addressed, the program jumps to the effective address; otherwise, the program continues at location P + 2. The sense lines can be configured in combination with the interrupt lines to permit more than one device to share an interrupt line. All DATA 620/i peripheral equipment include the sense lines required.

EXTERNAL CONTROL LINES

Discrete control lines are available as options in sets of eight. Each control line has a unique address. Up to 512 control lines can be addressed. The external control instruction is a one word instruction that places a pulse on the addressed control line. These are general purpose control lines that can be used to perform external control functions throughout a system. The control pulse has a 450-nanosecond width. The control lines required by DATA 620/i options are provided with the option.

PARALLEL I/O CHANNELS

The usual system application requires special devices to be connected to the computer. These devices can be interfaced with the computer in many ways. The system designer can implement the interface with his own electronics, purchase and assemble the appropriate logic modules (Micro-VersaLOGIC), or utilize the Varian Data Machine interface controllers.

The interface controllers provide the timing, gating and selection logic needed to communicate with the party line I/O lines under program control. The four available controllers are:

Gated inputed channel – provides a level input to the DATA 620/i party line

Gated output channel – provides a pulsed output from the DATA 620/i party line

Buffered input channel – provides an 18-bit register to receive pulsed inputs for subsequent input to the party line

Buffered output channel – provides 18 stored logic levels (flip-flops) for level output from the party line.

All four controllers are 18-bit parallel (on the 16-bit computer, 2 bits are not used) and greatly alleviate the interface problem.

PERIPHERAL EQUIPMENT

A full line of compatible peripheral equipment is available for the DATA 620/i series. Each device has been selected to meet the functional requirements of a real-time data system.

Each piece of peripheral equipment is provided with a controller that includes a party line adapter, buffering and control lines. The line printer, disc storage, and magnetic tapes include word assembly/disassembly registers. The magnetic tape control units contain double buffers to permit multiple simultaneous high-performance magnetic tape operation.

The peripherals will operate with the party line under program control, or automatically with an (optional) buffer interlace controller.

A complete line of analog conversion equipment is offered on a custom basis according to the requirement.

DATA 620/i SERIES PERIPHERAL EQUIPMENT

MAGNETIC TAPE SYSTEMS	Tape Controllers - Master controller for up to four tape transports. Will control 7 or 9 track transport and includes assembly/disassembly register.
	Tape Transports - Speeds of 45, 75, and 120 ips Densities of 200, 556, and 800 bpi. Seven and nine track industry compatible units.
AUXILIARY STORAGE	Fixed head rotating memory systems with capacities from 34K words to 500K words. Access times of 8.5 and 17 milliseconds. Transfer rates from 60 to 120 KC.
READERS AND PUNCHES	Card Reader - 1000 cpm Paper Tape Reader - 300 cps Paper Tape Punch - 60 and 120 cps
DIGITAL INPUT/OUTPUT	KEYBOARD ASR 33 Teletypewriter ASR 35 Teletypewriter KSR 35 Teletypewriter

GRAPHIC DEVICES	Oscilloscope Displays High Speed Printers - 300 and 600 LPM Electrostatic Plotters Digital Plotters - 300 steps per sec
MODEM INTERFACES	103, 201, and 301 types

SYSTEM SOFTWARE

A comprehensive package of operational programs are available with the DATA 620/i. These include a symbolic assembler, FORTRAN compiler, library of mathematical subroutines, debugging package, and a modular maintenance diagnostic package. The complete software package operates in the basic 8,192 words of core memory. In addition, Varian Data Machines has developed many real-time programs for a specific customer application. The more important portions of the Varian Data Machine software library are described below.

SYMBOLIC ASSEMBLER

The DATA 620/i assembler system (DAS) is a two-pass assembler that assists in program preparation by allowing instructions, addresses, etc., to be specified in a straightforward and meaningful manner. DAS recognizes over 20 pseudo-operations that aid the user in coding and debugging problems. Although DAS operates in a minimum system consisting of 4,096 words of core memory, paper tape reader, paper tape punch and typewriter, provisions have been made to utilize additional memory and peripheral equipment available to the system. Extensive syntax checking is performed during both passes of the assembler.

FORTRAN

DATA 620/i FORTRAN conforms with the proposed American standards for basic FORTRAN as published by the American Standards Association. The DATA 620/i FORTRAN, a one-pass complier, can operate in a 8,192 word computer equipped with only a model ASR-33 teletypewriter. Naturally, if higher performance peripherals are on the system, DATA 620/i FORTRAN utilizes them to produce faster compilation.

AID

AID is a collection of useful diagnostic and utility routines for the DATA 620/i computer. With this package, the programmer can call upon a wide variety of functions to aid him in debugging and running his programs. AID includes routines to correct memory, establish breakpoints, search memory, print memory, etc.

Also included in the AID package is a comprehensive binary paper tape handler that is particularly useful in preserving programs modified on the computer. This routine uses a standard address, data, and checksum format that is used by the DAS assembler.

DIAGNOSTIC PROGRAM PACKAGE

The DATA 620/i diagnostic program package is designed to check instructions, memory, and input/output devices, and to isolate errors. It can be used in either the preventative or the corrective mode of operation. In the preventative mode, the complete system is checked for operational readiness. If a malfunction exists, in most cases, the preventative will isolate the error. The corrective mode of operation is used when a malfunction is known to exist and the preventive mode does not decisively show the trouble. Proper application of these diagnostic routines can cut the mean-time-to-repair to minutes. This modular package can be easily expanded to accommodate any special system hardware tests.

SUBROUTINE LIBRARY

This comprehensive library includes the most commonly used subroutines needed in a systems environment. The library includes routines for logarithmic exponential and trigonometric functions, for fixed and floating-point arithmetic, and for operating standard peripheral equipment. Conventions and instructions are provided so the user can add application programs to the library and be called by DAS, FORTRAN and AID.

USER SERVICES

The purchase of a DATA 620/i includes support services designed to provide the user with start-up and sustaining service.

DOCUMENTATION

The documentation is comprehensive and clear, and contains the information required for the user to fully understand, program, operate and maintain the system. Interface and installation manuals are provided to the user prior to installation for system integration preparation. The program and service manuals are provided in advance of the user training attendance. The software manuals contain a special section covering software modularity and expansion techniques.

PROGRAMMING TRAINING*

Programming training courses are provided on a scheduled basis at Varian Data Machine facilities. The one week course covers instruction for programming in machine language, an introduction to the DATA 620/i software, and machine operation. The course includes time at the console. Supplies required for the course are provided at no charge to the attendees. On-site courses are available on a contract basis.

MAINTENANCE TRAINING*

A two-week at-the-factory maintenance course is provided on a scheduled basis. The instruction covers machine organization, operation, logic, design, timing, preventive maintenance, trouble-shooting, and repair. Extended training covering special systems hardware is available on an individual customer basis. The course is designed for personnel with existing digital logic design knowledge.

USER ORGANIZATION

Varian Data Machine Customer Services (CS) provide continuing coordination, program exchange and library maintenance for DATA 620 and DATA 620/i users. Users are notified of new additions to the library, application data, program and hardware modifications and new equipment. CS maintains up-to-date master prints on each system controlled. An inventory of programming forms, paper tapes and spare parts is maintained for expedited or emergency service. Statistical data on field operating experience based on user-submitted reports is maintained and available to users. On-call and on-site maintenance services are available on a contract basis.

*Available at nominal cost.

APPLICATION PROGRAMMING

Varian Data Machines' technical staff includes senior application programming specialists well-qualified to assist the user in the preparation of application programs. This professional group can assume full responsibility on a contract basis for the preparation of a total solution, including hardware and application programs.

DATA 620/i SPECIFICATIONS

TYPE	A system computer, general purpose digital, designed for on-line data system requirements, magnetic core memory, binary, parallel, single-address, with bus organization and micro-control.	
MEMORY	Magnetic core, 16 bits (18 bits optional), 1.8 microseconds full cycle, 700-nanoseconds access time, 4096 words minimum expandable to 32,768 words.	
ARITHMETIC	Parallel, binary, fixed point, 2's complement.	
WORD LENGTH	16 bits standard; 18 bits optional.	
SPEED (fetch and execute)	Add or Subtract	3.6 microseconds.
	Multiply (optional)	18.0 microseconds, 16-bit. 19.8 microseconds, 18-bit.
	Divide (optional)	18.0 to 25 microseconds, 16-bit. 19.8 to 28.8 microseconds, 18-bit.
	Register change class	1.8 microseconds.
	Input/Output - from A or B	3.6 microseconds.
	from memory	5.4 microseconds.
OPERATION REGISTERS	A register - accumulator, input/output, 16/18 bits. B register - double length accumulator, input/output, index register, 16/18 bits. X register - index register, 16/18 bits. P register - program counter, 16/18 bits.	
BUFFER REGISTERS	R register - operand register, 16/18 bits. U register - instruction register, 16/18 bits. S register - shift register, 5 bits, operates with the U register for executing shift instructions. L register - memory address register. W register - memory word register, 16/18 bits.	
CONTROL	Addressing modes: Direct addressing to 2,048 words. Relative to P register 512 words. Index with X register, hardware, does not add to execution time.	

Index with B register, hardware, does not add to execution time.
Multi-level indirect addressing.
Immediate.
Extended addressing (optional).

Instruction types:
- Single word.
- Double word.
- Generic.
- Micro-command.

Instructions: Over 100 standard commands, listed below, plus more than 128 macro-instructions:
- 3 load.
- 3 store.
- 5 arithmetic (2 optional).
- 3 logical.
- 10 jump.
- 10 jump and mark.
- 10 execute
- 14 immediate (2 optional).
- 13 input/output.
- 26 register change.
- 6 logical shift.
- 6 arithmetic shift.
- 2 control.
- 14 extended addressing (optional).
- Over 128 micro-instructions.

Micro-exec (optional):
Facility and hardware to construct a hardware program external to the DATA 620/i. Eliminates stored program memory accessing by use of hardware program.

Console:
Display and data entry switches for all operational registers, 3 sense switches, instruction repeat, single step; run; power on/off.

INPUT/OUTPUT

Processor input/output options:
Programmed data transfer:
- Single word to/from memory.
- Single word to/from A and B registers.
- External control lines.
- External sense lines.

Automatic Data Transfer:
Direct memory access facility transfer with rates over 200,000 words per second.

Priority Interrupts.
Group enable/disable, individually arm/disarm, single instruction interrupt capability.

Real-time clock:
Adjustable time base: May be programmed as multiple internal timers.

Power failure detect/restart:
Interrupts on power failure and automatically restarts on power recovery.

PHYSICAL

Dimensions:
Mainframe - 10-1/2 inches high, 19 inches wide, 15 inches deep

Weight:
Mainframe - 35 pounds.

Power:
3 amps 115vac, 60 Hz (340 watts). 115 ±10v, 60 ± 2 Hz. Power supplies are regulated. Additional regulation is not required under normal commercial power sources.
Conversion for 50 Hz and other voltages available at added cost.

Expansion:
Main processor contains provisions and space for all internal options.

Installation:
Mounts in standard 19-inch cabinet, no air conditioning, sub-flooring or special wiring and site preparation required.

Environments:
0°C to 45°C; 0% to 90% relative humidity.

MAINFRAME LOGIC AND SIGNALS

Integrated circuit, 8.8 MHz clock, logic levels 0v false, +5v true.

FULLY COMPATIBLE SYSTEM COMPONENTS

To increase your total system capability, Varian Data Machines offers a complete line of high-performance integrated circuit logic modules, small high-speed core memories and large mainframe memories for I/O equipment or additional system requirements. All have been field-proven with the DATA 620/i system, and are fully compatible with its power supply, voltage levels and signal requirements.

Micro-VersaLOGIC INTEGRATED CIRCUIT LOGIC MODULES

Micro-VersaLOGIC 5 MHz general purpose IC modules with NAND/NOR logic, and wired OR capacity at the collector, 5v logic levels, and excellent noise rejection over 1v. Over 25 module types, including universal flip-flops, delay multivibrators, clock drivers, 2-, 3- and 4-input expandable gates, and PNP to NPN interface modules. Compatible mounting hardware, including card files and card drawers, is also available.

VersaSTORE CORE MEMORIES

New high-speed core memory systems with integrated circuits and all-silicon components for highest reliability that operate asynchronously at 1.7 microseconds, with 750-nanosecond access time. VersaSTORE memories are available in increments up to 4,096 words of 36 bits, require only 5-1/4 inches of rack space, and can also be provided as 8k word memories of up to 18 bits.

Options include party line, built-in self-test, and a variety of timing and control flags.

VersaSTORE MAINFRAME MEMORIES

High-reliability VersaSTORE mainframe memories in sizes up to 65k words in 4k increments, with word lengths to 36 or 72 bits. Features include PNP to NPN interface, flexible input levels of 3v to 12v, continuous lamp display of address and data registers, servoed current drive, 2 μsec operation, integrated circuit design, and DATAGUARD protection system.

SYSTEM REFERENCE

SECTION I

INTRODUCTION

1.1 THE DATA 620/i

The DATA 620/i is a high-speed, parallel, binary computer. Its flexible design and modular packaging make it ideal for operation both as a general-purpose machine and for application as an on-line system component.

Its features include:

- Fast operation: 1.8-microsecond memory cycle.
- Large instruction repertoire: 107 standard, 18 optional; over 128 additional instruction configurations which can be micro-coded.
- Expandable word length: 16- or 18-bit configurations.
- Modular memory: 4096 word minimum, 32 768 maximum.
- Multiple addressing modes: direct, indirect, relative, index, immediate, and extended (optional).
- Flexible I/O: up to 64 devices on the I/O system, including optional interlaced data transmission and direct memory access operations.
- Extensive software: complete package includes an assembler, mathematice and I/O library, AID diagnostics, and an ASA FORTRAN subset.
- Modular packaging: mounts in a standard 19-inch cabinet. No special mechanical or environmental facilities are required.

The advance design techniques used throughout the DATA 620/i system provide solutions to real-time data acquisition, telemetry processing, process control, and simulation problems. In addition, the DATA 620/i is equally well suited for scientific computations. Special attention has been given to the interfacing problems usually encountered in integrating a digital computer into a system. As a result, the DATA 620/i can be joined to a system with unparalleled efficiency.

The unique design of the DATA 620/i makes it easy to program, operate and maintain. The entire mainframe includes the processor, all processor options, and a 4096-word core memory in a convenient 10-1/2 inch high rack-mountable package. Only 17 circuit boards, of 11 different types are used in the basic 16-bit configuration.

Power supplies for the processor and up to 8192 words of core memory are a separate 10-1/2 inch high rack-mountable package that mounts behind the mainframe. Thus, the entire computer requires only 10-1/2 inches of a standard 19-inch rack. Installation is easy, requiring no special mounting, cabling, or air conditioning provisions.

Maintainability of the DATA 620/i is enhanced by easy front access to all wiring, making it unnecessary to remove panels on the computer rack, obtain access to the modules, connectors, and wiring.

A complete set of software provided with the DATA 620/i permits rapid preparation of application programs. The system software includes:

- FORTRAN - Subset of ASA FORTRAN.
- DATA 620/i ASSEMBLY SYSTEM (DAS) - Two-pass symbolic assembler.
- AID - On-line debugging and utility package.
- MAINTAIN - Complete set of computer and peripheral diagnostics.
- SUBROUTINE LIBRARY - Complete library of transcendental functions, single- and double-precision and floating-point arithmetic, format conversion, and peripheral service routines.

A wide variety of peripheral equipments are available to provide the DATA 620/i user with a complete system suited to specific needs.

1.2 USE OF THE MANUAL

This manual provides the basic information required for programming and using the DATA 620/i, and is intended to be used in conjunction with other publications for the 620-series computers. These publications are listed in table 1-1.

The interface reference manual provides detailed information for installing the DATA 620/i, and for integrating the DATA 620/i with special system components.

Information required by the programmer for using the system software packages is contained in the programming reference, FORTRAN, and subroutine manuals.

The maintenance manuals contain the detailed design theory, logic and timing diagrams, circuit board data, maintenance procedures, and diagnostic programs.

Detailed design and maintenance information on peripheral device controllers is contained in individual reference manuals for these units. Operating and maintenance

Table 1-1
DATA 620/i DOCUMENTS

PUBLICATION NUMBER	TITLE
VDM-3000	System Reference Manual
VDM-3001	Interface Reference Manual
VDM-3002	Programming Reference Manual
VDM-3003	FORTRAN Manual
VDM-3004	Subroutine Manual
VDM-3005	Maintenance Manuals
VDM-3006	ASR-33 Teletype Controller Reference Manual
VDM-3007	Buffer Interlace Controller Reference Manual
VDM-3008	Magnetic Tape Controller Reference Manual
VDM-3009	600 LPM Line Printer Controller Reference Manual
VDM-3010	300 LPM Line Printer Controller Reference Manual
VDM-3011	Paper Tape System Controller Reference Manual
VDM-3012	100 CPM Card Reader Controller Reference Manual
VDM-3013	Priority Interrupt Reference Manual
VDM-3014	A/D Converter Reference Manual
VDM-3015	Optical Scanner Controller Manual
VDM-3016	ASR-35 Teletype Controller Reference Manual
VDM-3017	Digital Plotter Controller Reference
VDM-3018	DDC Disc Controller Reference Manual
VDM-3019	Console Printer Controller Reference Manual

procedures for optional peripheral devices (tape transports, printers, etc) are contained in the manufacturers' reference manuals furnished with the equipment.

Section II of this manual contains an overall description of the DATA 620/i system, and describes the word formats used in the computer. Section III describes the complete instruction set for the central processor. The input/output system, including all input/output, sense, control, and interrupt instructions is described in section IV. Section 5 provides information required for using the control console of the computer. Standard peripheral devices are described in section VI.

1.3 SPECIFICATIONS

Specifications of the DATA 620/i computer are listed in table 1-2.

Table 1-2
DATA 620/i SPECIFICATIONS

SPECIFICATION	CHARACTERISTICS
TYPE	General-purpose digital computer for on-line data system applications. Magnetic core memory: binary, parallel, single-address, with bus organization.
MEMORY	Magnetic core 16 bits (18 bits optional); 1.8 microseconds full-cycle, 700 nanoseconds access time, 4096 words minimum, expandable in 4096-word modules to 32,768 words. Power failure protection optional, non-volatile. Thermal overload protection is standard.
ARITHMETIC	Parallel, binary, fixed point, 2's complement.
WORD LENGTH	16 bits standard; 18 bits optional.
SPEED (fetch and execute)	
Add or Subtract	3.6 microseconds.
Multiply (optional)	16 bits - 18.0 microseconds. 18 bits - 19.8 microseconds.

Table 1-2 (continued)
DATA 620/i SPECIFICATIONS

SPECIFICATION	CHARACTERISTICS
Divide (optional)	16 bits - 18.0 to 25.2 microseconds. 18 bits - 19.8 to 28.8 microseconds.
Register Change	1.8 microseconds.
Input/Output	From A or B register - 3.6 microseconds. From memory - 5.4 microseconds.
OPERATIONAL REGISTERS	
A Register	Accumulator, input/output; 16 or 18 bits.
B Register	Low-order accumulator, input/output, index register; 16 or 18 bits.
X Register	Index register, multi-purpose register, 16 or 18 bits.
P Register	Instruction counter; 16 or 18 bits.
BUFFER REGISTERS	
R Register	Operand register, 16 or 18 bits.
U Register	Instruction register, 16 or 18 bits.
L Register	Memory location register, 12 bits.
W Register	Memory word register, 16 or 18 bits.
S Register	Shift register, 5 bits.
CONTROL	
Addressing Modes	Six as follows: Direct: to 2048 words.

Table 1-2 (continued)
DATA 620/i SPECIFICATIONS

SPECIFICATION	CHARACTERISTICS
	Relative to P register: to 512 words.
	Index with X register hardware: to 32,768 words (does not add to execution time).
	Index with B register, hardware: to 32,768 words (does not add to execution time).
	Multi-level indirect: to 32,768 words.
	Immediate: operand immediately follows instruction.
	Extended: operand address immediately follows instruction (optional).
Instruction Types	Four, as follows:
	Single word, addressing.
	Single word, non-addressing.
	Double word, addressing.
	Double word, non-addressing.
Instructions	107 standard, over 128 micro-instructions, plus 18 optional.
Micro-Exec (Option)	Facility and hardware to construct a hardwired program external to the DATA 620/i. Eliminates stored program memory accessing for hardwired programs.
Control Panel	Selectable display and data entry switches, three sense switches, instruction repeat, single step, run, power on/off, system reset.
	and

Table 1-2 (continued)
DATA 620/i SPECIFICATIONS

SPECIFICATION	CHARACTERISTICS
INPUT/OUTPUT	
Data Transfer	Three types as follows:
	Single word to/from memory (program control).
	Single word to/from A and B Registers (program control).
	Optional interlaced data channel (up to 202,000 words/second).
External Control (Select)	Up to 512 external control lines.
Program Sense	Up to 512 status lines may be sensed.
Interrupts	Power failure, thermal overload, (expandable in groups of eight) priority on/off, arm, disarm. Each interrupt line is associated with a unique memory.
PHYSICAL CHARACTERISTICS	
Dimensions	10-1/2 inches high x 13 inches deep.
Weight	90 pounds including power supplies.
Power	360 watts, single phase, 115 v ± 10 v, 47-440 Hz. Power supplies are regulated. Additional regulation is not required with normal commercial power sources.
Expansion	Mainframe package contains a 4096-word memory, the processor, and space for processor options. Additional memory requires an additional 10-1/2 inches of rack space for up to 12,288 words of

Table 1-2 (continued)
DATA 620/i SPECIFICATIONS

SPECIFICATION	CHARACTERISTICS
	additional storage. Peripheral controllers are mounted external to the mainframe.
Installation	Mainframe and power supply packages mount in 10-1/2 inches of standard 19-inch racks. No air-conditioning, subflooring, special wiring, or site preparation is required.
Environment	10° C to 45° C, 10% to 90% relative humidity.
LOGIC AND SIGNALS	The logic of the computer utilizes DTL and TTL integrated circuits employing 5 v levels. The logic levels on the transmission busses (I/O bus, interrupt bus, etc.) are reduced to 3 v to reduce cross talk and current requirements. Internal logic conventions are 5 v for logical 1 and 0 v for logical 0. Logic conventions on the busses is 3 v for logical 0, and 0 v for logical 1.
SOFTWARE	
DAS Assembler	Modular two-pass symbolic assembler which operates within the basic 4096-word memory. It includes 16 basic pseudo-ops. The 8192-word memory version includes over 30 pseudo-ops for programming ease.
FORTRAN	Modular one-pass compiler; subset of ASA FORTRAN for 8192-word memory.
AID	Program analysis package which assists programmers in operating the machine and debugging other programs. Includes basic operational executive subroutines.
MAINTAIN	Modular, two-mode diagnostic package which provides fast verification of central processor and

Table 1-2 (continued)
DATA 620/i SPECIFICATIONS

SPECIFICATION	CHARACTERISTICS
	peripheral operation, and assistance in isolating and correcting suspected faults.
Subroutines	Complete library of basic mathematical, fixed- and floating-point, single- and double-precision, number conversion and peripheral communication subroutines plus provisions for adding application-oriented routines.

SECTION II

DATA 620/i SYSTEM DESCRIPTION

2.1 COMPUTER ORGANIZATION

The DATA 620/i is organized with a unique bus structure, selection logic, and eight registers. The organization provides universal information routing, buffered processing, micro-programming capability, indexing without time penalty, and buffered input/output data transfer. A unique optional facility, Micro-EXEC, is also available which permits complex algorithms to be implemented with external control hardware. This capability provides increases in processing speed in excess of 400 percent over normal programmed operations.

The organization of the DATA 620/i is shown in figure 2-1. This diagram shows the major functional elements of the machine, including the registers and busses provided for information transfer.

The major functional elements of the DATA 620/i, indicated in figure 2-1, are: memory, control section, arithmetic/logic section, operational registers, internal busses, and input/output (I/O) bus.

2.1.1 Memory

The internal storage of the computer consists of 4096-word modules connected to the L and W busses. The mainframe can accommodate one 4096-word module. Additional modules are added in an additional frame that is attached to the mainframe. The computer memory can be expanded to a maximum of 32,768 words using 4096-word modules.

Instruction words read from memory are transferred to the control section for execution. Words may be transferred, under program control, from memory to the arithmetic/logic section, to the operational registers, or to the I/O bus. Words may be transferred, under program control, to memory from the operational registers or the I/O bus.

When one or more optional buffer interlace controler (BIC) is used, the system is capable of direct transfer between memory and peripheral devices on the I/O bus, concurrent with computations.

2.1.2 Control Section

The control section provides the timing and control signals required to perform all operations in the computer. The major elements in the section are the U register, the timing and decoding logic, and the shift control.

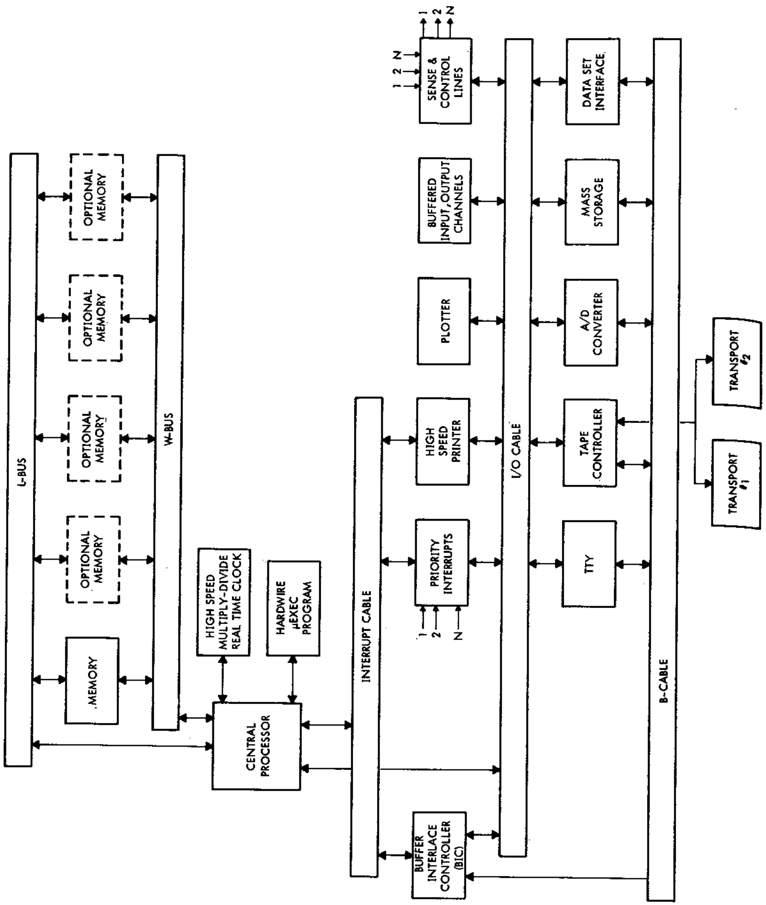


Figure 2-1. DATA 620/i Functional Organization

The U register (instruction register) is 16 bits long. This register receives each instruction from memory through the W bus and holds the instruction during its execution. The control fields of the instruction word are routed to the decoding and timing logic where the codes determine the required timing and control signals. The address field from U, used for various addressing operations, is also routed to the arithmetic/logic section.

The decoding logic decodes the fields of the instruction word held in U to determine the control signal levels required to perform the operations specified by the instruction. These levels select the timing signals generated by the timing unit.

Timing logic generates the basic 2.2-MHz system clock. From this clock, timing logic derives the timing pulses which control the sequence of all operations in the computer.

The shift control contains the shift counter and logic which control operations performed by the shift, multiply, and divide instructions.

2.1.3 Arithmetic/Logic Section

This section consists of two elements; the R register and the arithmetic unit.

The R register receives operands from memory and holds them during instruction execution. The operand may be either data or address words. This register permits transfers between memory and I/O bus during the execution of extended-cycle instructions.

The arithmetic unit contains gating required for all arithmetic, logic, and shifting operations performed by the computer. Indexed and relative address modifications are performed in this section without increased instruction execution time.

The arithmetic unit also controls the gating of words from the operational registers and the I/O bus onto the C bus where they are distributed to the operational registers or to memory registers. This facility is used to implement many of the micro-instructions of the computer.

2.1.4 Operational Registers

The basic DATA 620/i computer contains eight registers.

The operational registers consist of the A, B, X, and P registers. The A, B and X registers are directly accessible to the programmer. The P register is indirectly accessible through use of the jump class instructions which modify the program sequence. The operational registers are described in the following paragraphs.

A register. This full-length, 16/18-bit register is the upper half of the accumulator. This register accumulates the results of logical and addition/subtraction operations, the most-significant half of the double-length product in multiplication, and the remainder in division. It may also be used for input/output transfers under program control.

B register. This full-length, 16/18-bit register is the lower half of the accumulator. This register accumulates the least-significant half of the double-length product in multiplication, and the quotient in division. It may also be used for input/output transfers under program control and as a second hardware index register.

X register. This full-length 16/18-bit register permits indexing of operand addresses without adding time to execution of indexed instructions.

P register. This full-length, 16/18-bit register holds the address of the current instruction and is incremented before each new instruction is fetched. A full complement of instructions is available for conditional and unconditional modification of this register.

S register. This five-bit register controls the length of shift instructions in combination with the U register. This register also buffers memory from the control unit.

2.1.5 Internal Busses

C bus. This bus provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O bus, the operational registers, and the memory registers. The console display indicators are also driven from the C bus. Distribution of data simultaneously to multiple operational registers is facilitated by this bus.

S bus. This bus provides the parallel path and selection logic for routing data from the operational registers to the arithmetic unit.

W bus. The memory word (W) register is directly connected to all memory modules through the W bus. The bus is bidirectional and time-shared among memory modules.

L bus. The memory address (L) register is directly connected to all memory modules through the L bus. The bus is unidirectional.

2.1.6 Input/Output (I/O) Bus

The bidirectional I/O bus provides the parallel path between the computer and all peripheral devices. This bus contains the data and control lines required for transmitting ready, sense, function, and interrupt signals as well as data words between the computer and peripheral devices.

2.1.7 Direct Memory Access (DMA)

The DMA option allows data transfer into or out of memory modules without disturbing the contents of the operational registers. Only the L and W registers are altered. Access to memory using the DMA facility is on a "cycle-steal" basis and requires 2.7 microseconds of processor time per transfer.

2.1.8 Micro-EXEC*

The Micro-EXEC is a unique hardware technique for micro-step sequencing of the computer. This option provides hardware logic in which all computer control signals are made available on a pin board so that special hardware routines can be constructed. External control and special return instructions are provided for easy program entry and exit.

2.2 COMPUTER WORD FORMATS

There are three basic word formats used in the DATA 620/i: data, indirect address, and instruction. The instruction word format is further divided into four types: single-word addressing, single-word non-addressing, double-word addressing, and double-word non-addressing.

2.2.1 Data Word Format

The data word format is shown in figure 2-2. This word may be either 16 or 18 bits depending upon the word length configuration of a particular machine.

In the 16-bit format, the data occupies bit positions 0-14, with the sign in position 15. Negative numbers are represented in 2's-complement form. In the 18-bit format, the data occupies bits 0-16, with the sign in position 17.

2.2.2 Indirect Address Word Format

The indirect address word format is shown in figure 2-3. This word occupies a location in memory which is accessed by an instruction in the indirect address mode. Bit 15 contains the I Bit. If I = 0, bits 0-14 contain the location of an operand or instruction in memory. If I = 1, bits 0-14 contain the location of another indirect address word. Indirect addressing may be extended to any desired level. Each level of indirect addressing adds one cycle (1.8μs) to the basic execution time of an instruction.

2.2.3 Single-Word Instruction Formats

Single-word instructions may be either addressing or non-addressing, as defined in paragraphs 2.2.3.1 and 2.2.3.2.

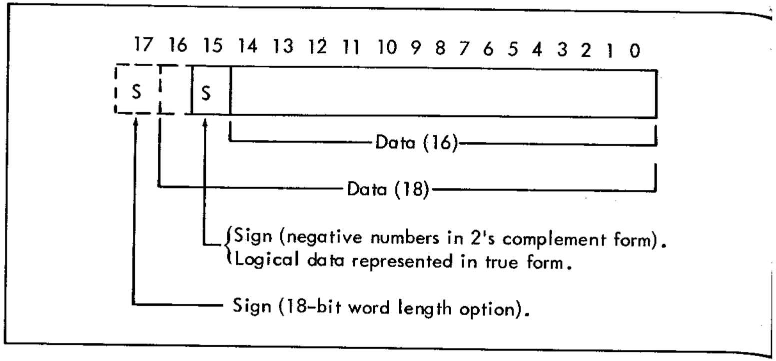


Fig. 2-2 Data Word Format

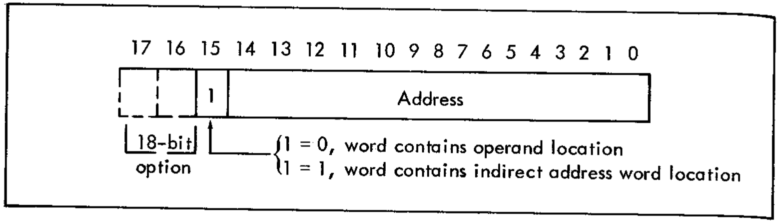


Fig. 2-3 Indirect Address Word Format

2.2.3.1 Addressing instructions. The single-word addressing instruction format is shown in figure 2-4. This type of word contains three fields, as follows:

o - Operation Code
m - Addressing Mode
a - Address Field

All single-word addressing instructions may be executed in any one of five addressing modes: direct, relative to P, index with X, index with B, and indirect.

Single-word addressing instruction groups are as follows:

LOAD/STORE
ARITHMETIC
LOGICAL

2.2.3.2 Non-addressing instructions. The single-word non-addressing instruction format is shown in figure 2-5. This instruction contains the following three fields:

c - Class Code
o - Operation Code
d - Definition

The d (definition field) specifies the action to be performed by the computer such as:

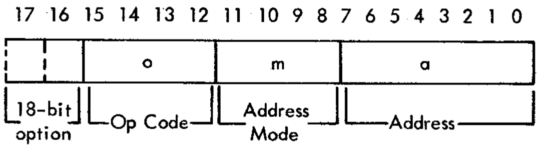
a. Number of shifts
b. Kind of register change as well as source and destination registers
c. Input/output
d. Halt code

Single-word non-addressing instruction groups are as follows:

SHIFT
CONTROL
REGISTER CHANGE
INPUT/OUTPUT

2.2.4 Double-Word Instruction Formats

Double-word instructions may be either addressing or non-addressing.



m Field:	0XX - Direct	operand in location 0 - 2047 (bits 10 to 0)
	100 - Relative	add a to P
	101 - Index (X)	add a to X
	110 - Index (B)	add a to B
	111 - Indirect	stored at a.

Fig. 2-4 Single-Word Addressing Instruction Format

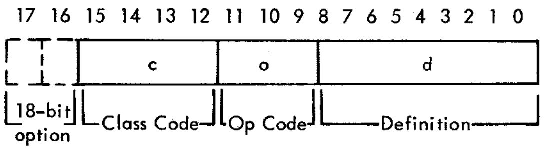


Fig. 2-5 Single-Word Non-Addressing Instruction Format

2.2.4.1 Addressing instructions. This instruction contains three fields:

c - Class Code
o - Operation Code
d - Definition

The double-word addressing instruction is shown in figure 2-6.

This format is used for the following instruction types:

JUMP
JUMP AND MARK
EXECUTE
EXTENDED ADDRESS

For the jump, jump and mark, and execute groups, the definition field of the first word defines a set of nine logical states which condition the execution of the instruction. The second word contains the jump address, jump-and-mark address, or the location of the instruction to be executed if the condition is met. Indirect addressing is permitted.

For the extended address group of instructions, the definition field is further divided into three subfields. The m field contains bits 0-2, the op code contains bits 3-6, with bits 7 and 8 left blank. Extended address instructions are identical in operation to the single-word addressing instructions except that they allow direct addressing to 32,768 words of memory.

For the memory input/output group, the definition field of the first word contains the number of the peripheral device and its mode, and the second word contains the memory address of the data to be transferred. Indirect addressing is permitted.

2.2.4.2 Non-addressing instructions. The double-word non-addressing instruction format is shown in figure 2-7. This format is used for the Immediate group of instructions. There are 12 standard and two optional instructions in this group.

The op code field contains the operation to be performed (bits 3-6). All single-word addressing type instructions may be performed as an immediate type instruction. The operand is contained in the second word. Indirect addressing is not applicable.

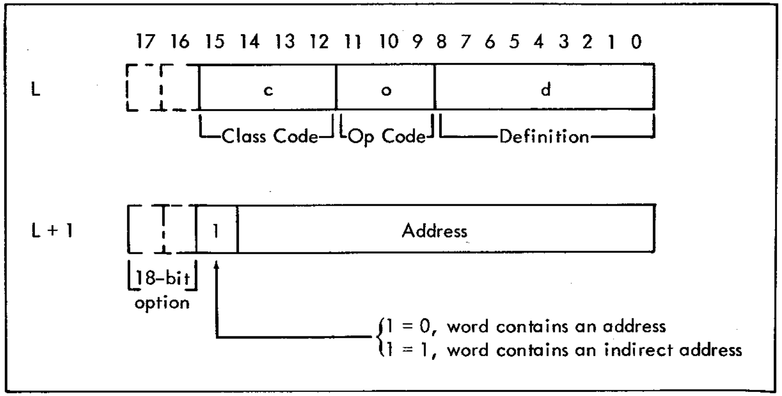


Fig. 2-6 Double-Word Addressing Instruction Format

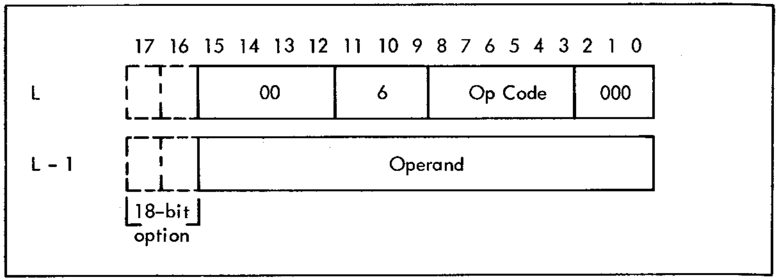


Fig. 2-7 Double-Word Instruction Format Immediate Type Instructions

SECTION III

DATA 620/i CENTRAL PROCESSOR INSTRUCTIONS

3.1 GENERAL

This section describes DATA 620/i instructions which affect operations in the central processor. Input/output instructions are described in section IV. Information provided for each instruction is as follows:

- The mnemonic that is recognized by the DATA 620/i assembler (DAS)
- Mnemonic definition
- Instruction timing
- Instruction description
- Registers altered by execution of the instruction
- Addressing modes permitted
- A flow chart, when required for complete understanding.

Instructions are divided into two classes: single-word and double-word. Each class contains both addressing and non-addressing groups of instructions. Microprogramming operations which can be implemented for various instruction types are summarized in appendix G.

3.2 SINGLE-WORD INSTRUCTIONS

Single-word instructions may be either addressing or non-addressing. The addressing instruction groups are:

LOAD/STORE
ARITHMETIC (multiply/divide optional)
LOGICAL

The non-addressing instruction groups are:

CONTROL
SHIFT
REGISTER CHANGE

3.2.1 Single-Word Addressing Instructions

The format of the single-word addressing class instructions is shown in figure 2-4. The operation is specified by the o field (bits 12-15). The address field, a (bits 0-8), contains the base location of an operand in memory. Operand addressing may be in any one of five modes specified by the m field (bits 9-11).

Table G1(d), appendix G, summarizes the addressing modes, and tables G1(a), G1(b), and G1(c) summarize the operation codes for the single-word addressing instructions. Figure 3-1 shows the general operand addressing flow for this class of instructions.

For direct addressing, bits 0-10 specify the location of an operand within the first 2048 (0-2047) words of memory.

For relative addressing, the address field is added to the P register, mod 2^9, to form the effective address. This mode permits addressing an operand up to 511 words in advance of the current program location.

For index addressing with the X register, the address field is added to the X register, mod 2^{15}, to form the effective address. Indexing does not increase the basic instruction execution time.

For index addressing with the B register, the address field is added to the B register, mod 2^{15}, to form the effective address. Indexing does not increase the basic instruction execution time.

For indirect addressing, the address field specifies the location of an indirect address word within the first 512 (0-511) words of memory. If I = 0 in the address word, the word contains the location of an operand. If I = 1, the word specifies the location of another indirect address word. Each level of indirect addressing adds one cycle (1.8 μs) to the basic instruction execution time.

3.2.1.1 Load/Store instruction group. The following paragraphs provide the nmemonic, description, and timing for each instruction in the load/store group. Figures 3-2 and 3-3 show the general flow for the load/store instruction group.

LDA Load A Register Timing: 2 cycles

17	16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
		01	m	a

(Bits 17–16: 18-bit option)

The contents of the addressed memory location are placed in the A register.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A

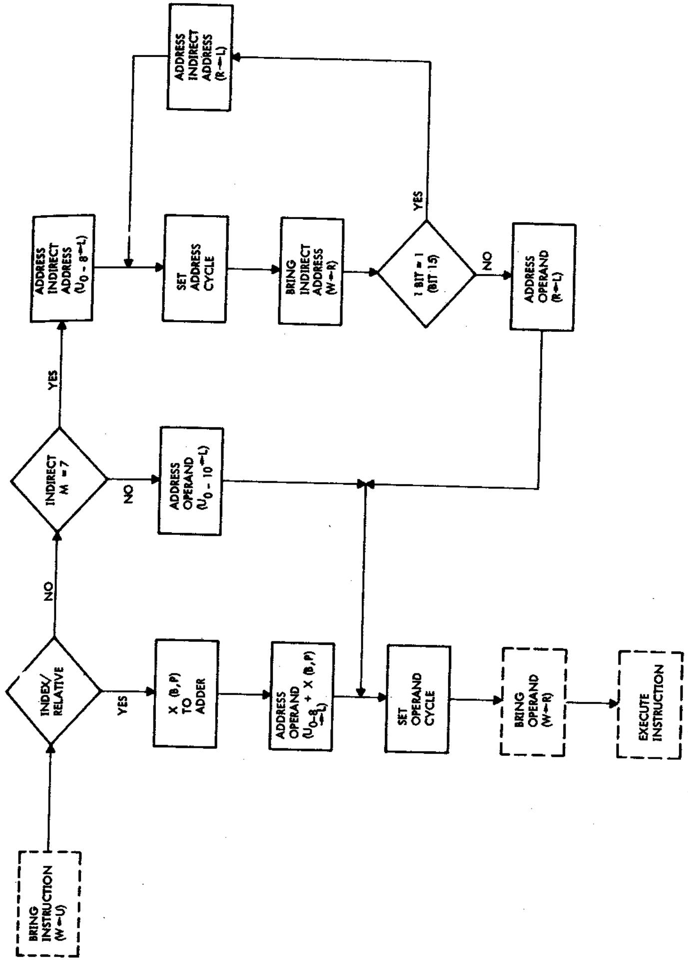


Figure 3-1. Single-Word Address Instruction, Operand Addressing, General Flow.

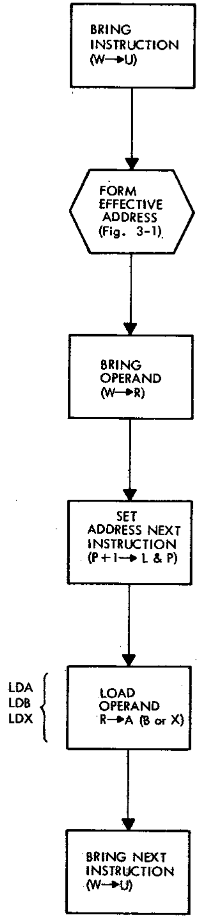


Figure 3-2. Load-Type Instruction, General Flow.

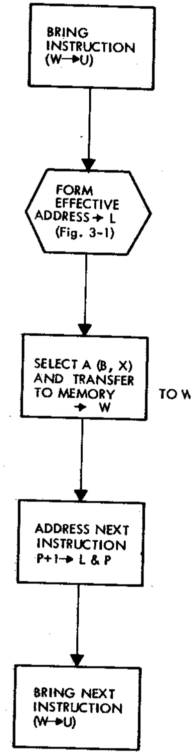


Figure 3-3. Store-Type Instruction, General Flow.

LDB — Load B Register — Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
	02	m	a

The contents of the effective memory location are placed in the B register.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: B

LDX — Load Index Register — Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
	03	m	a

The contents of the effective memory location are placed in the Index register.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: X

STA — Store A Register — Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
	05	m	a

The contents of the A register are placed in the effective memory location.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: Memory

STB — Store B Register — Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
	06	m	a

The contents of the B register are placed in the effective memory location.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: Memory

STX — Store Index Register — Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
	07	m	a

The contents of the b register are placed in the effective memory location

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: Memory

3.2.1.2 Arithmetic instruction group. The following paragraphs provide the mnemonic, description, and timing for each instruction in the arithmetic group. Figures 3-4 and 3-5 show the general flow for the arithmetic instruction group.

INR — Increment Memory and Replace — Timing: 3 cycles

17 16 (18-bit option)	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
	04	m	a

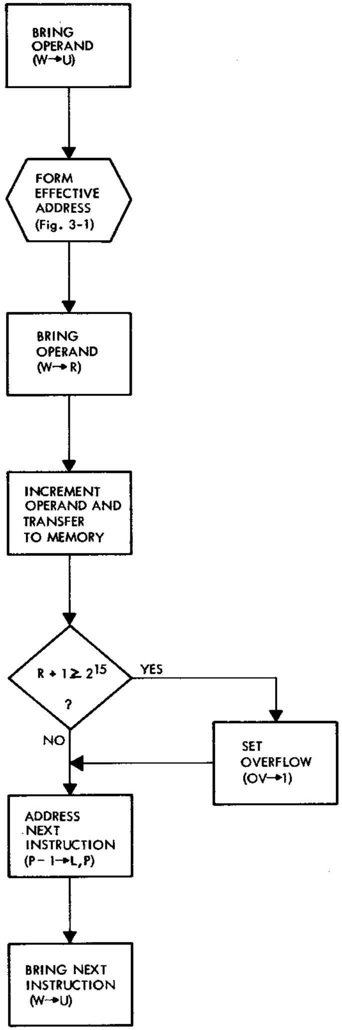


Figure 3-4. Increment Memory and Replace Instruction, General Flow.

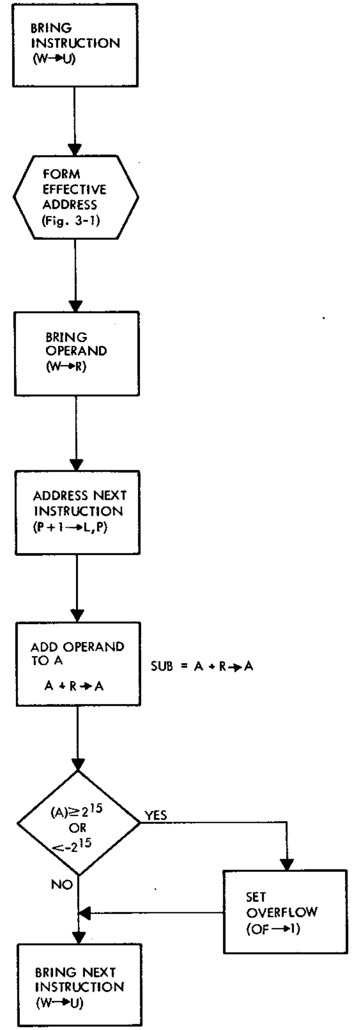


Figure 3-5. Add Instruction, General Flow.

The contents of the effective memory location are incremented by one, mod 2^{16} (2^{18}).

After execution, if (M) $\geq 2^{15}$ (2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: Memory, OF

ADD — Add Memory to A — Timing: 2 cycles

17	16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
18-bit option		12	m	a

The contents of the effective memory location are added to the contents of the A register and the sum is placed in the A register.

After execution, if (A) $\geq 2^{15}$ (2^{17}) or $< -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A, OF

SUB — Subtract Memory from A — Timing: 2 cycles

17	16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
18-bit option		14	m	a

The contents of the effective memory location are subtracted from the A register and the difference is placed in the A register.

After execution, if (A) $\leq 2^{15}$ (2^{17}) or $< -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A, OF

MUL — Multiply (optional) — Timing: 10 cycles (16 bits), 11 cycles (18 bits)

17	16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
18-bit option		16	m	a

The contents of the B register are multiplied by the contents of the effective memory location. The contents of the A register are added to the contents of the B register at the start of the operation. The product is placed in the A and B registers, with the most-significant half of the product in the A register and the least-significant half in the B register. The sign of the product is contained in the sign position of the A register. The sign position of the B register is set to "0".

The algorithm is in the form A · B(X) + A.

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A, B

DIV — Divide (Optional) — Timing: 10-14 cycles (16 bits), 11-16 cycles (18 bits)

17	16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
18-bit option		17	m	a

The contents of the A and B registers are divided by the contents of the effective memory location. The quotient is placed in the B register with sign, and the remainder is placed in the A register with the sign of the dividend.

If $$\frac{(A, B)}{M} \leq 1$$

(divisor ≥ dividend, taken as a binary fraction), overflow will not occur. If overflow does occur, the overflow indicator (OF) is set.

3.2.1.3 Logical instruction group. The following paragraphs provide the mnemonics, description, and timing for each instruction in the logical instruction group.

ØRA Inclusive-OR Memory and A Timing: 2 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
(17–16)	11	m	a

An inclusive-OR operation is performed between the effective memory location and the contents of the A register. The result is placed in the A register. If either the effective memory location or A contain a "1" in the same bit position, a "1" is placed in the result. The truth table is shown below:

OPERATION		RESULT
An	Effective Memory Location (n)	An
0	0	0
0	1	1
1	0	1
1	1	1

where n = bit position

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A

ERA Exclusive-OR Memory and A Timing: 2 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
(17–16)	13	m	a

An exclusive-OR operation is performed between the effective memory location and the contents of the A register. The result is placed in the A register. If the same bit position of the effective memory location and A contain a "0", or if both bit positions

contain a "1", the result is "0". If the same bit position of the effective memory location and A are not equal; i.e., one contains a "0" and the other a "1" the result is a "1". The truth table is shown below:

OPERATION		RESULT
An	Effective Memory Location (n)	An
0	0	0
0	1	1
1	0	1
1	1	0

where n = bit position

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A

ANA AND Memory and A Timing: 2 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
(17–16)	15	m	a

The logical-AND is performed between the contents of the A register and the contents of the effective memory location. The result is placed in the A register. If the same bit position of both the effective memory location and A contain a "1", the result is a "1". The truth table is shown below:

OPERATION		RESULT
An	Effective Memory Location (n)	An
0	0	0
0	1	0
1	0	0
1	1	1

where n = bit position

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: P

3.2.2 Single-Word Non-Addressing Instructions

The format of the single word non-addressing instruction class is shown in figure 2-5.

A non-addressing single-word instruction includes the control group, the shift group, and the register change group. The operation is defined by the m field. The address field (a), as such, is not used by the control group instructions. For the shift group, the a field defines the type and number of shifts. For the register change group, the a field defines the type of transfer and the registers affected.

3.2.2.1 Control instruction group. The following paragraphs provide the mnemonic, description, and timing for each instruction in the control group. Table G2, appendix G, summarizes the control instructions.

HLT Halt Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	0	XXX

When the computer executes the halt instruction, computation is stopped and the computer is placed in the STEP mode. When the RUN button is pressed, computation starts with the next instruction in sequence.

Indexing: No
Indirect Addressing: No
Registers Altered: None

NØP No Operation Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	5	000

Execution of the NØP instruction does not affect the A, B, X registers or memory.

Indexing: No
Indirect Addressing: No
Registers Altered: None

SØF Set Overflow Indicator Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	7	401

The overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: OF

RØF Reset Overflow Indicator Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	7	400

The overflow indicator (OF) is reset.

Indexing: No
Indirect Addressing: No
Registers Altered: OF

3.2.2.2 Shift instruction group. For shift instructions 0-31, the address field (a) defines the type of shift (bits 4-8) and the number of bit positions to be shifted (bits 0-4). The instruction format showing the use of each a-field bit is given in table G3(a), appendix G. Twelve of the possible sixteen shift operations defined by bits 4-8 are implemented. These are summarized in table G3(b). Figure 3-6 shows the general flow for the shift instructions.

LSRA Logical Shift A Right Timing: 1 + 0.25 n cycles (n = number of shifts)

17 16 15	14 13 12 11 10 9	8 7 6 5	4 3 2 1 0
18-bit option	00	4	340 + n

The contents of the A register are shifted n places to the right (n = 0 to 37_8). "0's" are shifted into the high-order positions of the A register. Information shifted out of the the low-order position of the A register is lost.

Indexing: No
Indirect Addressing: No
Registers Altered: A

LSRB Logical Shift B Right Timing: 1 + 0.25 n cycles (n = number of shifts)

17 16 15	14 13 12 11 10 9	8 7 6 5	4 3 2 1 0
18-bit option	00	4	140 + n

The contents of the B register are shifted n places to the right (n = 0 to 37_8). Information shifted out of the low-order position of the B register is lost. "0's" are shifted into the high-order position of the B register.

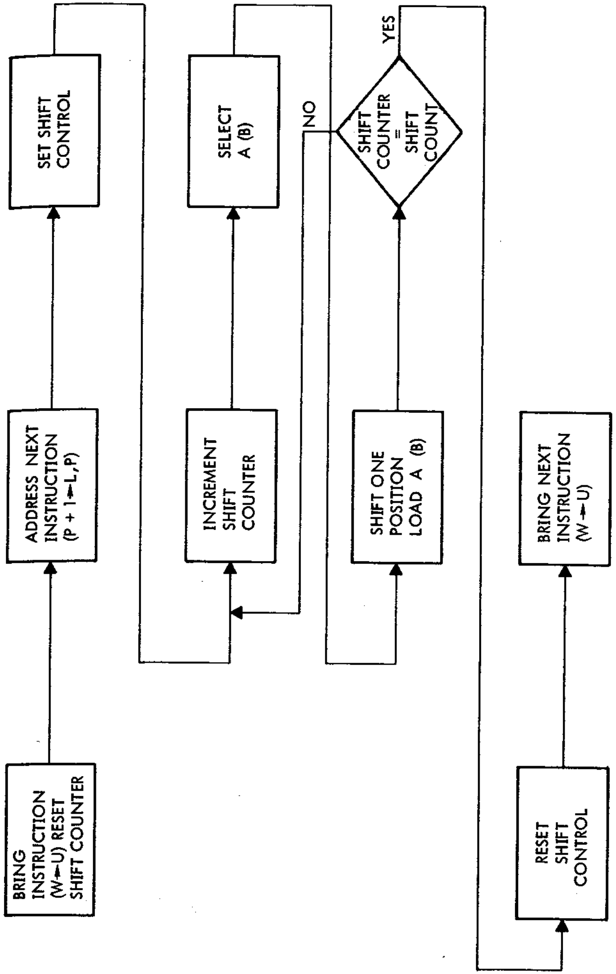
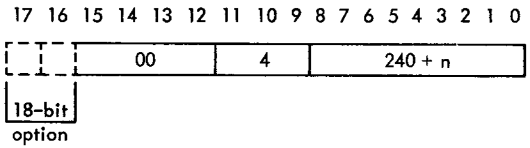


Figure 3-6. Single-Register Shift Instruction, General Flow.

Indexing: No
Indirect Addressing: No
Registers Altered: B

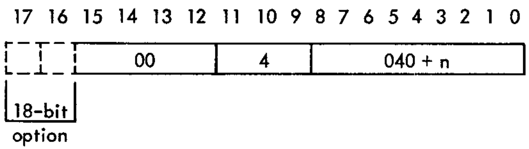
LRLA Logical Rotate A Left Timing: 1 + 0.25 n cycles
(n = number of shifts)



The contents of the A register are rotated left n places (n = 0 to 37_8). Bit position A_{15} (A_{17}) is rotated into bit position A_0.

Indexing: No
Indirect Addressing: No
Registers Altered: A

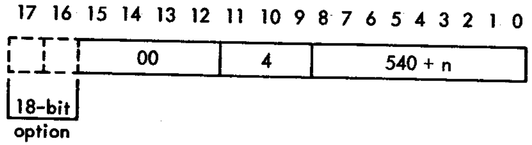
LRLB Logical Rotate B Left Timing: 1 + 0.25 n cycles
(n = number of shifts)



The contents of the B register are rotated n positions to the left (n = 0 to 37_8). Bit position B_{15} (B_{17}) is rotated into bit position B_0.

Indexing: No
Indirect Addressing: No
Registers Altered: B

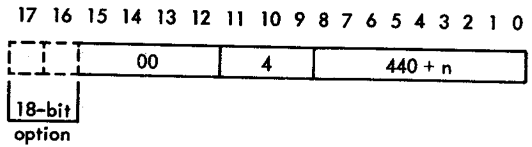
LLSR Long Logical Shift Right Timing: 1 + 0.50 n cycles
(n = number of shifts)



The contents of the A and B registers are shifted right n positions (n = 0 to 37_8). Bits shifted out of the low-order position of B are lost. "0's" are shifted into the high-order position of the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B

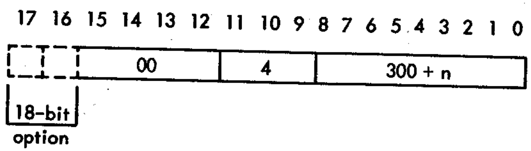
LLRL Long Logical Rotate Left Timing: 1 + 0.50 n cycles
(n = number of shifts)



The contents of the A and B registers are rotated n postions to the left (n = 0 to 37_8). Bit position A_{15} (A_{17}) is shifted into bit position B_0.

Indexing: No
Indirect Address: No
Registers Altered: A, B

ASRA Arithmetic Shift A Right Timing: 1 + 0.25 n cycles
(n = number of shifts)



The contents of the A register are shifted n positions to the right ($n = 0$ to 37_8). Bits shifted out of the low-order position of A are lost. The sign bit of A, A_{15} (A_{17}) is extended n places to the right.

Indexing: No
Indirect Addressing: No
Registers Altered: A

ASLA Arithmetic Shift A Left Timing: 1 + 0.25 n cycles (n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	4	200 + n

The contents of the A register are shifted n places to the left ($n = 0$ to 37_8). The sign bit, A_{15} (A_{17}), is retained and "0's" are shifted into the low-order positions of A. Bits shifted out of A_{14} (A_{16}) are lost.

Indexing: No
Indirect Addressing: No
Registers Altered: A

ASRB Arithmetic Shift B Right Timing: 1 + 0.25 n cycles (n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	4	100 + n

The contents of the B register are shifted n places to the right ($n = 0$ to 37_8). Information shifted out of the low-order position of B are lost. The sign bit of B, B_{15} (B_{17}) is extended n places to the right.

Indexing: No
Indirect Addressing: No
Register Altered: B

ASLB Arithmetic Shift B Left Timing: 1 + 0.25 n cycles (n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	4	000 + n

The contents of the B register are shifted n places to the left ($n = 0$ to 37_8). The sign bit of B, B_{15} (B_{17}), is retained and "0's" are shifted into the low-order positions of B. Bits shifted out of B_{14} (B_{16}) are lost.

Indexing: No
Indirect Addressing: No
Registers Altered: B

LASR Long Arithmetic Shift Right Timing: 1 + 0.50 n cycles (n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option			
	00	4	500 + n

The contents of the A and B registers are shifted n places to the right ($n = 0$ to 37_8). Bit position A_0 is shifted into bit position B_{14} (B_{16}). The sign of the A register, A_{15} (A_{17}), is extended n places to the right. The sign bit, B_{15} (B_{17}) of the B register remains unchanged. Bits shifted out of the low-order position of the B register are lost.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B

LASL Long Arithmetic Shift Left — Timing: 1 + 0.50 n cycles (n = number of shifts)

17 16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
18-bit option	00	4	400 + n

The contents of the A and B registers are shifted n places to the left (n = 0 to 37_8). Bit position B_{14} (B_{16}) is shifted into bit position A_0, with the sign of B, B_{15} (B_{17}) remaining unchanged. The sign of the A register, A_{15} (A_{17}) is not altered. Information shifted out of A_{14} (A_{16}) is lost and "0's" are shifted into the low-order positions of the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B

3.2.2.3 Register change group. The register change instruction group provides a macro-operation facility, in that these instructions may combine several register change operations in a single instruction. The instruction format is shown in figure 3-7.

The address field (a) defines the source and destination of a parallel word transfer within the operational register set A, B, and X. Any combination of registers may be selected. The a field also specifies whether the word transferred will be unchanged, incremented, decremented, or complemented. The transfer may also be conditional on the overflow indicator.

Table G4(a), in appendix G, defines the transfer control specified by the a field. If more than one source register is specified, the result will be the inclusive-OR of the group. Complementing causes transfer of the complement of the inclusive-OR (NOR) of a combination of source registers. A total of 512 different register change operations are possible. The most useful instructions are contained in the mnemonic repertoire recognized by the DAS assembler, summarized in table G4(b), appendix G.

IAR Increment A Register — Timing: 1 cycle

17 16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
18-bit option	00	5	111

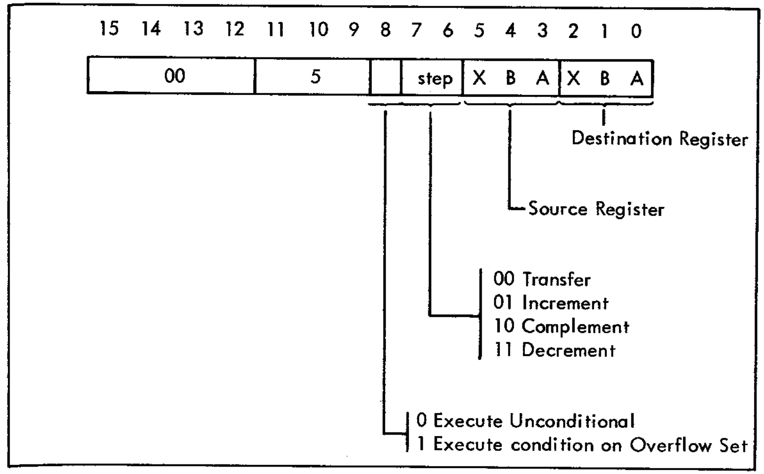


Fig. 3-7 Register Change Instruction

IBR Increment B Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	122

IXR Increment X Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	144

The contents of the A (B, X) register are incremented by one, mod 2^{16} (2^{18}). If the sign of the A (B, X) register changes from plus to minus, the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X), OF

DAR Decrement A Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	311

DBR Decrement B Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	322

DXR Decrement X Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	344

The contents of the A (B, X) register are decremented by one, mod 2^{16} (2^{18}). If the sign bit of the A (B, X) register is changed from minus to plus, the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X), OF

CPA Complement A Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	211

CPB Complement B Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	222

CPX Complement X Register Timing: 1 cycle

17–16	15–12	11–9	8–0
18-bit option	00	5	244

The contents of the A (B, X) register are complemented (1's-complement).

Indexing: No
Indirect Addressing: No
Register Altered: A (B, X)

TAB Transfer A Register to B Register Timing: 1 cycle

17	16	15–12	11–9	8–0
18-bit option		00	5	012

The contents of the A register are placed in the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

TAX Transfer A Register to X Register Timing: 1 cycle

17	16	15–12	11–9	8–0
18-bit option		00	5	014

The contents of the A register are placed in the X register.

Indexing: No
Indirect Addressing: No
Registers Altered: X

TBA Transfer B Register to A Register Timing: 1 cycle

17	16	15–12	11–9	8–0
18-bit option		00	5	021

The contents of the B register are placed in the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

TBX Transfer B Register to X Register Timing: 1 cycle

17	16	15–12	11–9	8–0
18-bit option		00	5	024

The contents of the B register are placed in the X register

Indexing: No
Indirect Addressing: No
Registers Altered: X

TXA Transfer X Register to A Register Timing: 1 cycle

17	16	15–12	11–9	8–0
18-bit option		00	5	041

The contents of the X register are placed in the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

TXB Transfer X Register to B Register Timing: 1 cycle

17	16	15–12	11–9	8–0
18-bit option		00	5	042

The contents of the X register are placed in the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

TZA Transfer Zero to A Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	001

TZB Transfer Zero to B Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	002

TZX Transfer Zero to X Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	004

The A (B, X) register is cleared to zero.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X)

AØFA Add Overflow to A Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	511

AØFB Add Overflow to B Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	522

AØFX Add Overflow to X Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	544

The contents of the overflow indicator (OF) are added to the A (B, X) register, mod 2^{16} (2^{18}). The sum is placed in the A (B, X) register. The overflow flip-flop does not change.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X)

SØFA Subtract Overflow from A Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	711

SØFB Subtract Overflow from B Register Timing: 1 cycle

17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		
18-bit option	00	5	722

SØFX	Subtract Overflow from X Register	Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

(18-bit option)	00	5	744

The contents of the overflow indicator (OF) are subtracted from the A (B, X) register, mod 2^{16} (2^{18}). The overflow flip-flop does not change.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X)

3.3 DOUBLE-WORD INSTRUCTIONS

Double-word instructions may be either addressing or non-addressing. The instructions of the double-word addressing group are:

JUMP
JUMP-AND-MARK
EXECUTE
EXTENDED ADDRESSING (optional)

The instruction in the double-word non-addressing group is:

IMMEDIATE

3.3.1 Double-Word Addressing Instructions

For double-word addressing instructions, the second word is contained in the memory location following the instruction word. The second word may contain an operand or an address. The address may be either indirect or direct. The general flow chart for double-word instructions is shown in figure 3-8.

Bits 0 through 8 determine the conditions for execution of the instruction. The condition is tested if the corresponding bit is equal to "1". For example, if bit 0 equals "1", the instruction will examine the status of the overflow flip-flop. If overflow is set, the command will be executed. If overflow is not set, the next instruction in sequence will be executed.

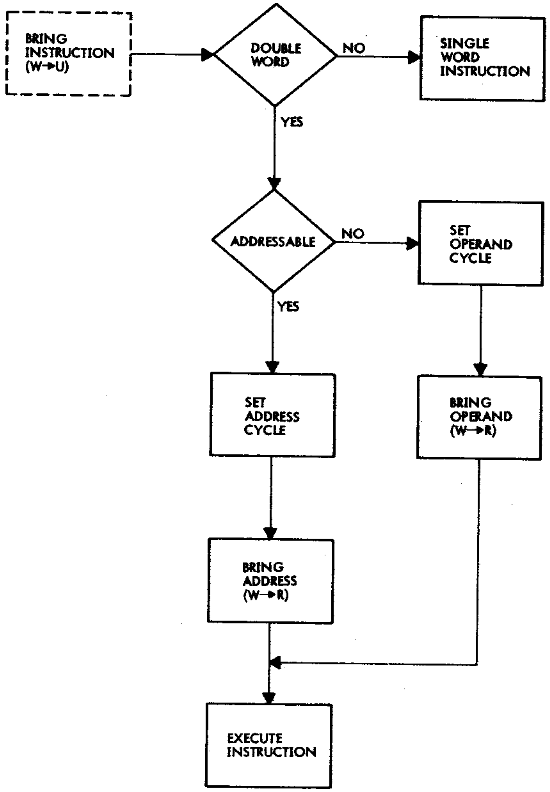


Figure 3-8. Double-Word Instruction, General Flow.

3.3.1.1 Jump instruction group. For the jump instruction group, the address field (a) contains a set of nine flags which define the logical conditions for execution of the jump function. The jump address is contained in the second word of the double-word instruction. Table G-5(a), in appendix G, summarizer the logical condition associated with each bit in the address field. The jump condition is the logical-AND of all "1's" in the field. Thus, there are 512 possible combinations, but not all are useful. The most useful conditional jump instructions are contained in the mnemonic instruction repertoire recognized by the DAS assembler, summarized in Table G-5(b). The general flow for jump instruction is shown in figure 3-9.

JMP Jump Unconditionally Timing: 2 cycles

	17 16 (18-bit option)	15 14	13	12 11 10 9 8 7 6 5 4 3 2 1 0
n		00	1	000
n+1		Jump Address		

The next instruction executed is at the jump address.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

JØF Jump if Overflow Indicator Set Timing: 2 cycles

	17 16 (18-bit option)	15 14	13	12 11 10 9 8 7 6 5 4 3 2 1 0
n		00	1	001
n+1		Jump Address		

If the overflow indicator (OF) is set, the next instruction executed is at the jump address. If the overflow indicator is not set, the next instruction in sequence is executed. The overflow indicator is reset upon execution of the JØF instruction.

Indexing: No
Indirect Addressing: Yes
Registers Altered: OF (reset), P

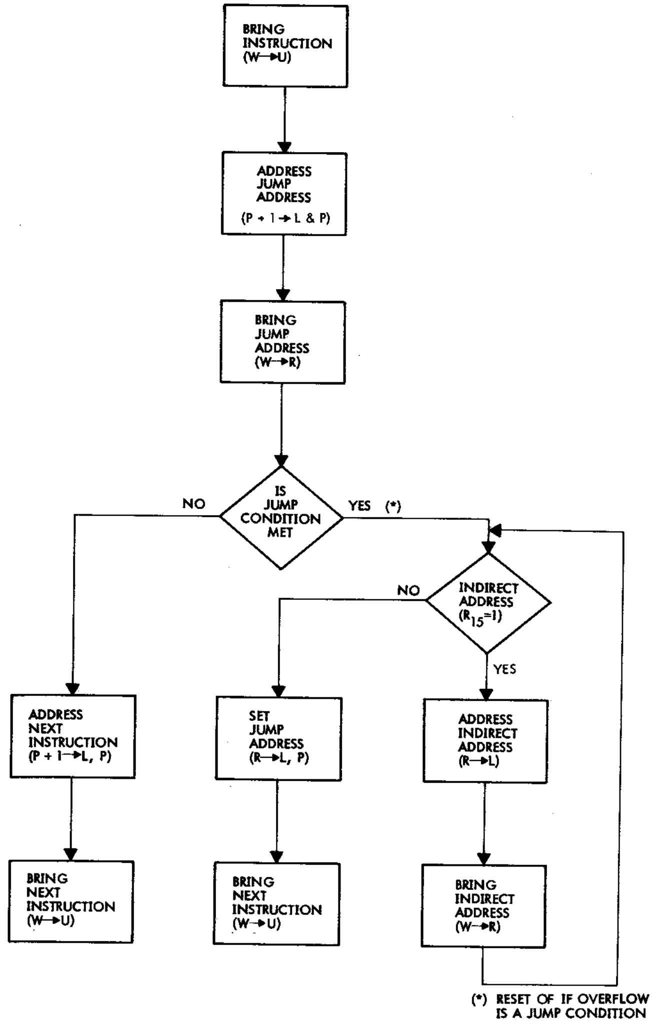
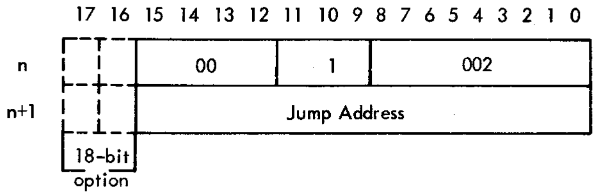


Figure 3-9. Jump Instruction, General Flow.

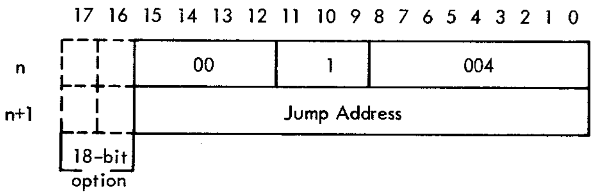
JAP Jump if A Register Positive Timing: 2 cycles



If the contents of the A register are positive or zero, the next instruction executed is at the jump address. If the A register is negative, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

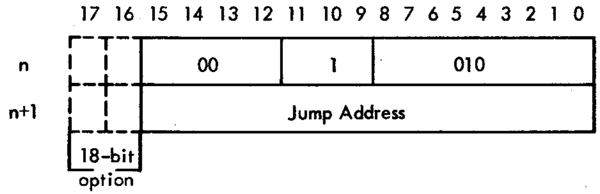
JAN Jump if A Register Negative Timing: 2 cycles



If the A register is negative, the next instruction executed is at the jump address. If the A register is positive, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

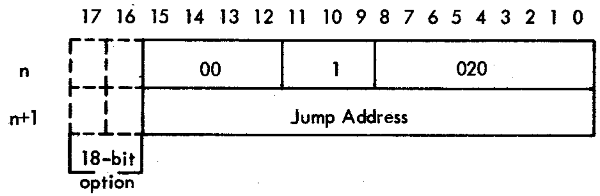
JAZ Jump if A Register Zero Timing: 2 cycles



If the A register is zero, the next instruction executed is at the jump address. If the A register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

JBZ Jump if B Register Zero Timing: 2 cycles



If the B register is zero, the next instruction executed is at the jump address. If the B register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

JXZ — Jump if X Register Zero — Timing: 2 cycles

	17	16	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
n			00	1	040
n+1			Jump Address		
	18-bit option				

If the index register (X) is zero, the next instruction executed is at the jump address. If the register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

JSS1 — Jump if Sense Switch 1 Set — Timing: 2 cycles

	17	16	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
n			00	1	100
n+1			Jump Address		
	18-bit option				

JSS2 — Jump if Sense Switch 2 Set — Timing: 2 cycles

	17	16	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
n			00	1	200
n+1			Jump Address		
	18-bit option				

JSS3 — Jump if Sense Switch 3 Set — Timing: 2 cycles

	17	16	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
n			00	1	400
n+1			Jump Address		
	18-bit option				

If sense switch 1 (2, 3) is set, the next instruction executed is at the jump address. If the sense switch being tested is not set, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

3.3.1.2 Jump and mark instruction group. For the jump and mark group of instructions, the address field a defines the same set of logical conditions specified for the jump group. These conditions are summarized in table G6(a) in appendix G. Thus, there are 512 possible combinations, but not all are useful. The most convenient instructions are contained in the mnemonic instruction repertoire recognized by the DAS assembler. These are summarized in table G6(b).

JMPM — Jump and Mark Unconditionally — Timing: 3 cycles

	17	16	15 14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
n			00	2	000
n+1			Jump Address		
	18-bit option				

The contents of the instruction counter (P) are stored at the jump address. The next instruction executed is at the jump address plus one.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

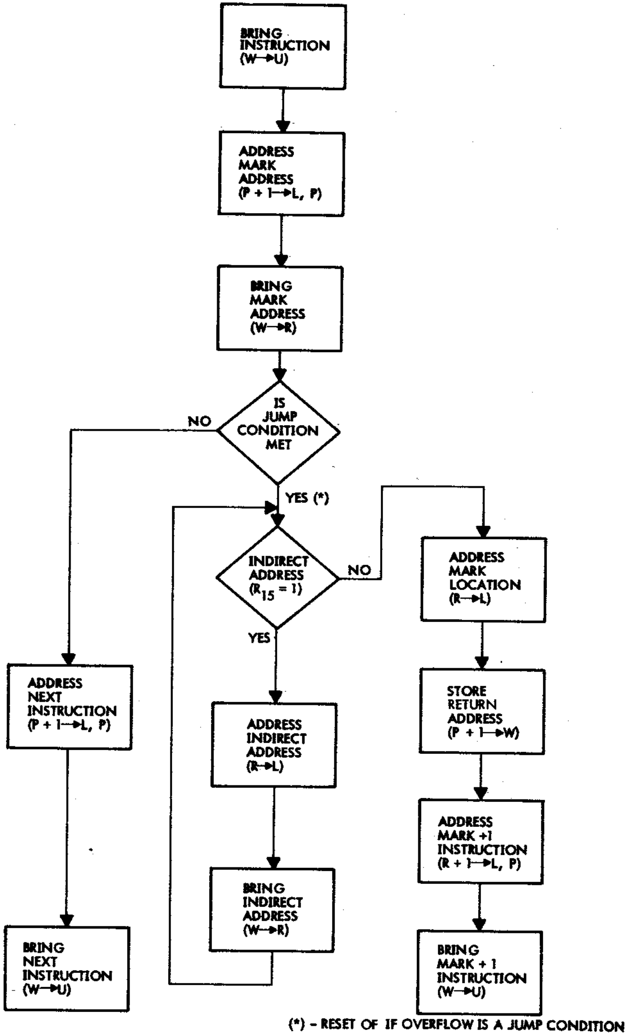


Figure 3-10. Jump-and-Mark Instruction, General Flow.

JØFM — Jump and Mark if Overflow Set — Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12 11 10	9 8 7	6 5 4 3 2 1 0
n		00	2	001
n+1		Jump Address		

If the overflow indicator (OF) is set, the contents of the instruction counter (P) are stored at the jump address, and the instruction at the jump address plus one is executed. If the overflow indicator is not set, the next instruction in sequence is executed. The overflow indicator is reset upon execution of the JØFM instruction.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P, OF (reset)

JANM — Jump and Mark if A Register Negative — Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12 11 10	9 8 7	6 5 4 3 2 1 0
n		00	2	004
n+1		Jump Address		

If the A register is negative, the contents of the instruction counter (P) are placed at the jump address, and the instruction at the jump address plus one is executed. If the A register is positive, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

JAPM Jump and Mark if A Register Positive Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	2	002
n+1		Jump Address		

If the A register is positive or zero, the contents of the instruction counter (P) are placed at the jump address, and the instruction at the jump address plus one is executed. If the A register is negative, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

JAZM Jump and Mark if A Register Zero Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	2	010
n+1		Jump Address		

If the A register is zero, the instruction counter (P) is placed at the jump address and the instruction at the jump address plus one is executed. If the A register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

JBZM Jump and Mark if B Register Zero Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	2	020
n+1		Jump Address		

If the B register is zero, the contents of the instruction counter (P) are placed at the jump address, and the instruction at the jump address plus one is executed. If the B register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

JXZM Jump and Mark if X Register Zero Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	2	040
n+1		Jump Address		

If the X register is zero, the contents of the instruction counter (P) are placed at the jump address and the instruction at the jump address plus one is executed. If the X register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

JS1M Jump and Mark if Sense Switch 1 Set Timing: 3 cycles

	17	16	15–12	11–9	8–0
			17 16 15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n			00	2	100
n+1			Jump Address		
	18-bit option				

JS2M Jump and Mark if Sense Switch 2 Set Timing: 3 cycles

	17	16	15–12	11–9	8–0
n			00	2	200
n+1			Jump Address		
	18-bit option				

JS3M Jump and Mark if Sense Switch 3 Set Timing: 3 cycles

	17	16	15–12	11–9	8–0
n			00	2	400
n+1			Jump Address		
	18-bit option				

If sense switch 1 (2, 3) is set, the instruction counter (P) is placed at the jump address, and the instruction at the jump address plus one is executed. If the tested sense switch is not set, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

3.3.1.3 Execute instruction group. For the execute group of instructions, the address field a contains a set of nine flags which define the logical conditions for executing an instruction contained at the effective execution address. The execution address is contained in the second word of the double-word instruction. Table G7(a), appendix G, summarizes the logical conditions associated with each bit in the address field. The execute condition is the logical-AND of all "1's" in the a field. The most useful of the 512 possible execute instructions are contained in the mnemonic instruction repertoire recognized by the DAS assembler, summarized in table G7(b). Figure 3-11 illustrates the general flow for the execute instructions.

It is important to note that only single-word instructions should be executed. The single-word instruction groups are:

LOAD/STORE
ARITHMETIC
LOGICAL
CONTROL
SHIFT
REGISTER CHANGE

If the execute is attempted on double-word instructions, erroneous operation will occur. The double-word instruction groups are:

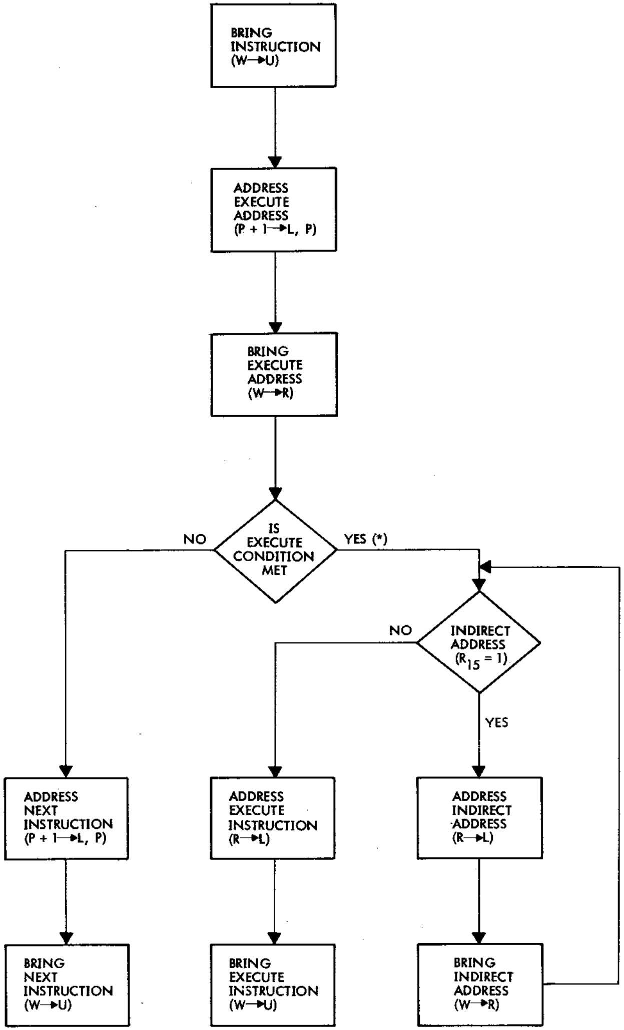
JUMP
JUMP AND MARK
EXECUTE
EXTENDED ADDRESSING (optional)
IMMEDIATE

XEC Execute Unconditionally Timing: 2 cycles

	17	16	15–12	11–9	8–0
n			00	3	000
n+1			Execute Address		
	18-bit option				

The instruction located at the execute address is executed and then the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: None



(*) RESET OF IF OVERFLOW WAS AN EXECUTE CONDITION

Figure 3-11. Execute Instruction, General Flow.

XØF Execute if Overflow Set Timing: 2 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	3	001
n+1		Execute Address		

If the overflow indicator (OF) is set, the instruction at the execute address is executed, and then the next instruction in sequence is executed.

If the overflow indicator is not set, the next instruction in sequence is executed. Execution of the XØF instruction resets the overflow indicator.

Indexing: No
Indirect Addressing: Yes
Registers Altered: OF (reset)

XAP Execute if A Register Positive Timing: 2 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	3	002
n+1		Execute Address		

If the A register is positive or zero, the instruction at execut address is executed, and then the next instruction in sequnece is executed. If the A register is negative, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: None

XAN

Execute if A Register Negative Timing: 2 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	3	004
n+1		Execute Address		

If the A register is negative, the instruction at the execute address is executed, and then the next instruction in sequence is executed. If the A register is positive, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: None

XAZ

Execute if A Register Zero Timing: 2 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	3	010
n+1		Execute Address		

If the A register is zero, the instruction at the execute address is executed, and then the next instruction in sequence is executed.

If the A register is not zero the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: None

XBZ

Execute if B Register Zero Timing: 2 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	3	020
n+1		Execute Address		

If the B register is zero, the instruction at the execute address is executed, and then the next instruction in sequence is executed.

If the B register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: None

XXZ

Execute if X Register Zero Timing: 2 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		00	3	040
n+1		Execute Address		

If the index register (x) is zero, the instruction at the execute address is executed, and then the next instruction in sequence is executed.

If the index register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Register Altered: None

XSI

Execute if Sense Switch 1 — Timing: 2 cycles

	17 16 (18-bit option)	15 14	13 12 11 10	9 8 7 6 5 4 3 2 1 0
n		00	3	100
n+1		Execute Address		

XS2

Execute if Sense Switch 2 — Timing: 2 cycles

	17 16 (18-bit option)	15 14	13 12 11 10	9 8 7 6 5 4 3 2 1 0
n		00	3	200
n+1		Execute Address		

XS3

Execute if Sense Switch 3 — Timing: 2 cycles

	17 16 (18-bit option)	15 14	13 12 11 10	9 8 7 6 5 4 3 2 1 0
n		00	3	400
n+1		Execute Address		

If sense switch 1, (2, 3) is set, the instruction at the execute address is executed and then the next instruction in the sequence is executed. If the sense switch tested is not set, the next instruction is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: None

3.3.1.4 Extended addressing instruction group (optional). The extended address mode instructions are similar in format to the Immediate Instructions. However, the second word of the double-word instruction contains the effective address. The address can be indirect or direct. It is determined by bit 15 of the second word.

U_{15} … U_{12}	U_{11} … U_9	U_8 … U_3	U_2 … U_0
00	6	YY	X

OP CODE ADDRESS MODE

YY equals any single word instruction in the op code.

If X =	ADDRESS MODE	EFFECTIVE ADDRESS
0 - 3	Immediate	Second word contains operand
4	Relative to P	Contents of second word + (P register + 1)
5	Indexed with X	Contents of second word +X register
6	Indexed with B	Contents of second word + B register
7	Direct or indirect	Contents of second word is the direct address if bit 15 is "0". Contents of second word is an indirect address if bit 15 is "1".

LDAE

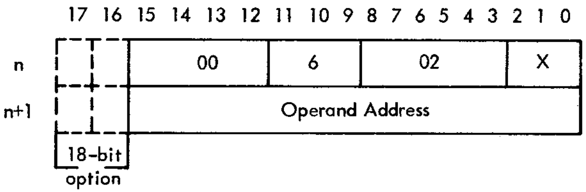
Load A Register Extended (optional) — Timing: 3 cycles

	17 16 (18-bit option)	15 14	13 12 11	10 9 8 7 6 5 4 3	2 1 0
n		00	6	01	X
n+1		Operand Address			

The contents of the memory location as addressed by the operand address at location n + 1 are placed in the A register.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A

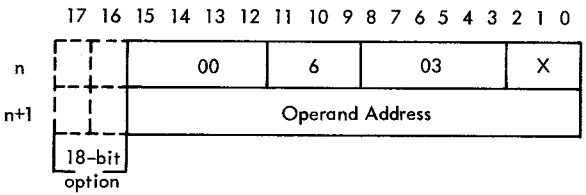
LDBE   Load B Register Extended (optional)   Timing: 3 cycles



The contents of the memory location as addressed by the operand address at location n + 1 are placed in the B register.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: B

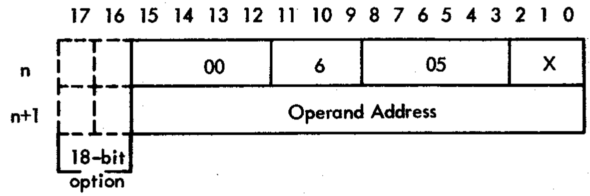
LDXE   Load X Register Extended (optional)   Timing: 3 cycles



The contents of the memory location as addressed by the operand address at location n + 1 are placed in the X register.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: X

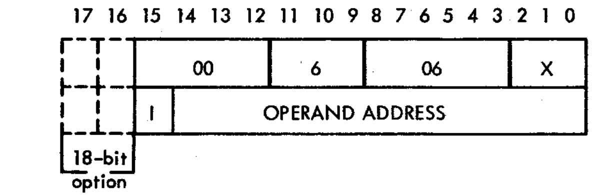
STAE   Store A Register Extended (optional)   Timing: 3 cycles



The contents of the A register are stored in the memory location as addressed by the operand address at location n + 1.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory

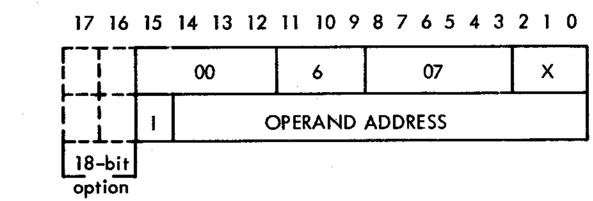
STBE   Store B Register Extended (optional)   Timing: 3 cycles



The contents of the B register are stored in the memory location as addressed by the operand address to location n + 1.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory

STXE   Store Index Register Extended (optional)   Timing: 3 cycles



The contents of the index register are stored in the memory location as addressed by the operand address at location n + 1.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory

INRE Increment Memory and Replace Extended (optional) Timing: 4 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option				
	00	6	04	X
	I	OPERAND ADDRESS		

The contents of the memory location as addressed by the operand address at location n + 1 are incremented by one, mod 2^{16} (2^{18}).

After execution, if (M) $\geq 2^{15}$ (2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory, OF

ADDE Add Memory to A Extended (Optional) Timing: 3 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option				
	00	6	12	X
	I	OPERAND ADDRESS		

The contents of the memory location as addressed by the operand address at location n + 1 are added to the contents of the A register and the sum is placed in the A register.

After execution, if (A) $\geq 2^{15}$ (2^{17}) or $< -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A, OF

SUBE Subtract Memory from A Extended (optional) Timing: 3 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

18-bit option				
	00	6	14	X
	I	OPERAND ADDRESS		

The contents of the memory location as addressed by the operand address at location n + 1 are subtracted from the contents of the A register and the difference is placed in the A register.

After execution, if (A) $\geq 2^{15}$ (2^{17}) or $< -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A, OF

MULE Multiply Extended (optional) Timing: 11 cycles (16 bits)
12 cycles (18 bits)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

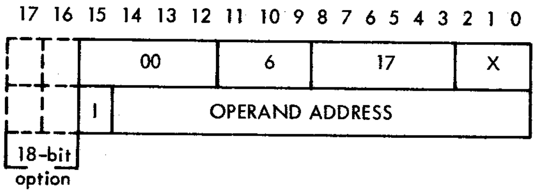
18-bit option				
	00	6	16	X
	I	OPERAND ADDRESS		

The contents of the B register are multiplied by the contents of the memory location as addressed by the operand address in location n + 1. The contents of the A register are added to the contents of the B register at the start of the operation. The product is placed in the A and B registers with the most-significant half of the product in the A register and the least-significant half in the B register. The sign of the product is contained in the sign position of the A register. The sign position of the B register is set to "0".

The algorithm is in the form A · B(X) + A.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A, B

DIVE Divide Extended (optional) Timing: 11-15 cycles (16 bits)
12-17 cycles (18 bits)



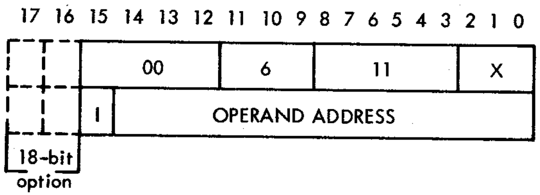
The contents of the A and B registers are divided by the contents of the memory location as addressed by the operand address at location n + 1. The quotient is placed in the B register and the remainder is placed in the A register.

If $$\frac{(A, B)}{M} \leq 1$$

(divisor ≥ dividend, taken as a binary fraction), overflow will not occur. If overflow does occur, the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A, B, OF

ØRAE Inclusive-OR Memory and A Extended (optional) Timing: 3 cycles



The inclusive-OR operation is performed between the contents of the A register and the contents of the memory location as addressed by the operand address in location n + 1.

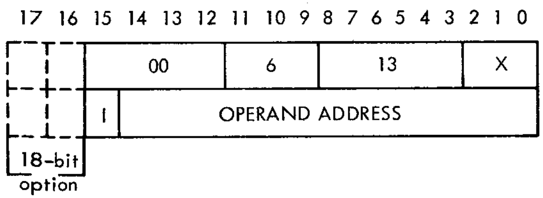
The result is placed in the A register. If either the memory location or A contain a "1" in the same position, a "1" is placed in the result. The truth table is shown below:

OPERATION		RESULT
An	Effective Memory Location (n)	An
0	0	0
0	1	1
1	0	1
1	1	1

Where n = bit position

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A

ERAE Exclusive-OR Memory and A Extended (optional) Timing: 3 cycles



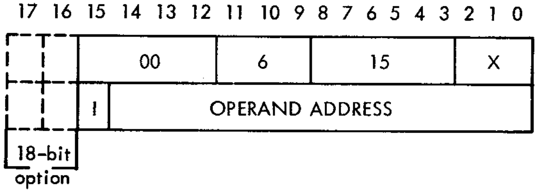
An exclusive-OR operation is performed between the contents of the A register and the contents of the memory location as addressed by the operand address in location n + 1. The result is placed in the A register. If the same bit position of the memory location and the A register contain a "0", or if both bit positions contain a "1", the result is "0". The turth table is shown below:

OPERATION		RESULT
An	Effective Memory Location (n)	An
0	0	0
0	1	1
1	0	1
1	1	0

Where n = bit position

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A

ANAE — AND Memory and A Extended (optional) — Timing: 3 cycles



The logical-AND operation is performed between the contents of the A register and the contents of the memory location as addressed by the operand address in location n + 1. The result is placed in the A register. If the same bit position of both the memory location and the A register contain a "1" the result is a "1". The truth table is shown below:

OPERATION		RESULT
An	Effective Memory Location (n)	An
0	0	0
0	1	0
1	0	0
1	1	1

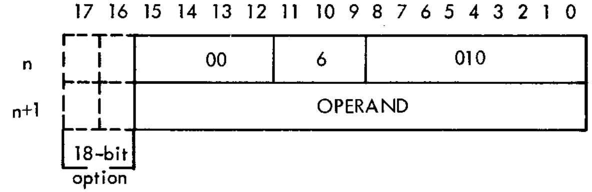
Where n = bit position

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A

3.3.2 Double-Word Non-Addressing Instructions

The double-word non-addressing instructions consist of the Immediate instruction group. The operand for the immediate instruction is contained in the second word of the double-word instruction. Address modification is not permitted for this group of intructions. The immediate instruction group codes are summarized in table G10, appendix G.

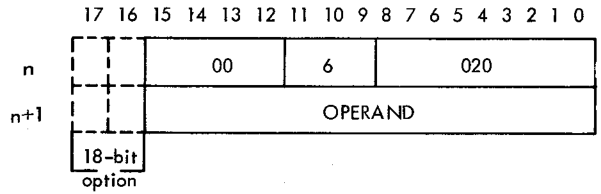
LDAI — Load A Register Immediate — Timing: 2 cycles



The contents of the operand at location n + 1 are placed in the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

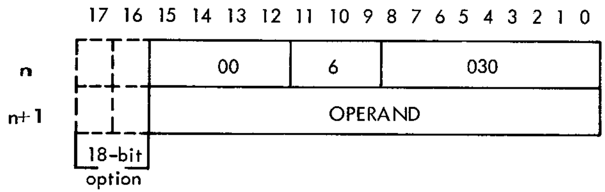
LDBI — Load B Register Immediate — Timing: 2 cycles



The contents of the operand at location n + 1 are placed in the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

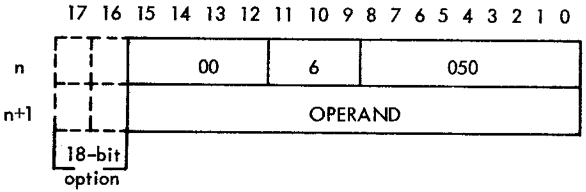
LDXI — Load X Register Immediate — Timing: 2 cycles



The contents of the operand at location n + 1 are placed in the X register.

Indexing: No
Indirect Addressing: No
Registers Altered: X

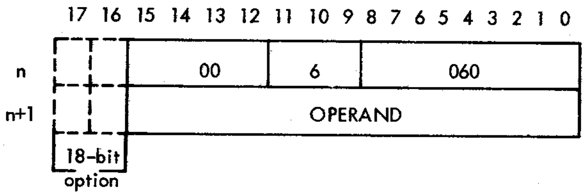
STAI Store A Register Immediate Timing: 2 cycles



The contents of the A register are placed in the operand at location n + 1.

Indexing: No
Indirect Addressing: No
Registers Altered: Operand

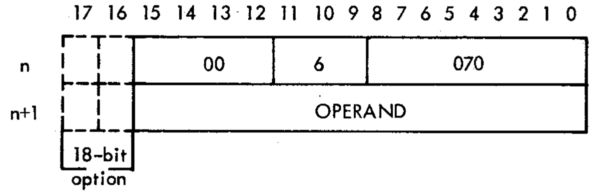
STBI Store B Register Immediate Timing: 2 cycles



The contents of the B register are placed in the operand at location n + 1.

Indexing: No
Indirect Addressing: No
Registers Altered: Operand

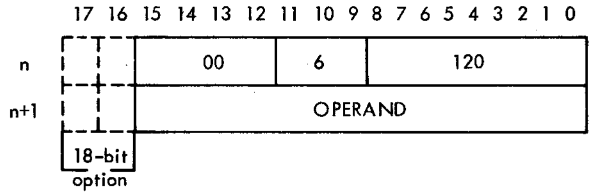
STXI Store X Register Immediate Timing: 2 cycles



The contents of the Index register are placed in the operand at location n + 1.

Indexing: No
Indirect Addressing: No
Registers Altered: Operand

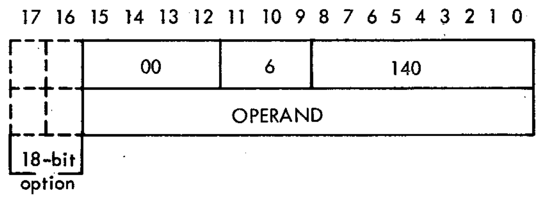
ADDI Add Immediate Timing: 2 cycles



The contents of the A register are added to the contents of the operand at location n + 1. The sum is placed in the a register. After execution, if (A) $\geq 2^{15}$ (2^{17}) or $< -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: A, OF

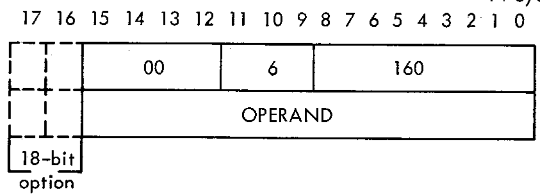
SUBI Subtract Immediate Timing: 2 cycles



The contents of the operand at location n + 1 are subtracted from the contents of the A register. The difference is placed in the A register. After execution, if (A) ≥ 2^{15} (2^{17}) or < -2^{15} (-2^{17}), the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: A, OF

MULI — Multiply Immediate (optional) — Timing: 10 cycles (16 Bits), 14 cycles (18 Bits)

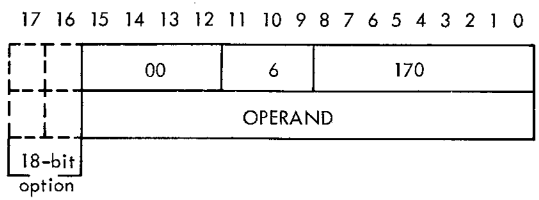


The contents of the B register are multiplied by the contents of the operand at location n + 1. The contents of the A register are added to the contents of the B register at the start of the operation. The product is placed in the A and B registers, with the most-significant half of the product in the A register and the least-significant half in the B register. The sign of the product is contained in the sign position of the A register. The sign position of the B register is set to "0".

The algorithm is in the form A · B(X) + A.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B

DIVI — Divide Immediate (optional) — Timing: 10-14 cycles (16 bits), 11-16 cycles (18 bits)



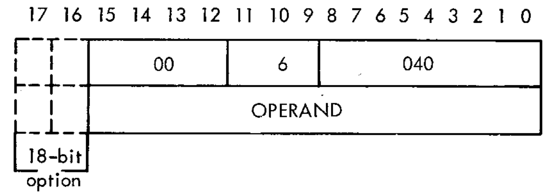
The contents of the A and B registers are divided by the contents of the operand at location n + 1. The quotient is placed in the B register with sign, and the remainder is placed in the A register with the sign of the dividend.

If $$\frac{(A, B)}{M} \leq 1$$

(divisor ≥ dividend, taken as a binary fraction), overflow will not occur. If overflow does occur, the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B, OF

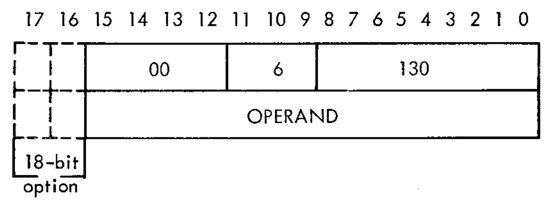
INRI — Increment and Replace Immediate — Timing: 3 cycles



The contents of the operand at location n + 1 are incremented by one, mod 2^{16} (2^{18}). After execution, if (n + 1) ≥ 2^{15} (2^{17}), the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: Operand, OF

ERAI — Exclusive-OR Immediate — Timing: 2 cycles



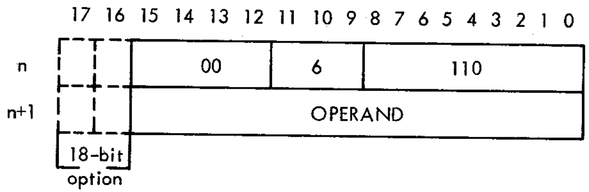
An exclusive-OR is performed between the contents of the operand at location n + 1 and the contents of the A register, and the result is placed in the A register. If the same bit position of the operand and the A register contain a "0", or if both bit positions contain a "1", the result is set to "0". The truth table is shown below:

OPERAND		RESULT
An	OPERAND(n)	An
0	0	0
0	1	1
1	0	1
1	1	0

where n = bit position

Indexing: No
Indirect Addressing: No
Registers Altered: A

ØRAI   Inclusive-OR Immediate   Timing: 2 cycles



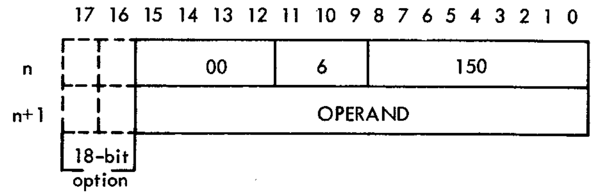
An inclusive-OR is performed between the contents of the operand and the contents of the A register. The result is placed in the A register. If either the operand or the A register contains a "1" in the same bit position, a "1" is placed in the result in the A register. The truth table is shown on the following page.

OPERAND		RESULT
An	OPERAND(n)	An
0	0	0
0	1	1
1	0	1
1	1	1

where n = bit position

Indexing: No
Indirect Addressing: No
Registers Altered: A

ANAI   AND Immediate   Timing: 2 cycles



A logical-AND is performed between the contents of the operand and the contents of the A register. The result is placed in the A register. If the same bit position of the operand and the A register contain a "1", the result is set to "1"; otherwise, the result is set to "0". The truth table is shown below:

OPERATION		RESULT
An	OPERAND(n)	An
0	0	0
0	1	0
1	0	0
1	1	1

where n = bit position

Indexing: No
Indirect Addressing: No
Registers Altered: A

SECTION IV

DATA 620/i INPUT/OUTPUT SYSTEM

4.1 INTRODUCTION

This section describes the operation and instruction set of the computer input/output system which includes the data transfer, external control, program sense, and program interrupt facilities.

The DATA 620/i input/output system is designed to facilitate integration of the computer into an overall system. Refer to the interface reference manual for detailed information required for special interface designs.

A wide selection of optional peripheral devices is also available.

4.2 ORGANIZATION

As shown in the block diagram, figure 2-1, the I/O section of the computer communicates with the operational registers and the memory through the internal C bus. Data and control signals are transmitted to and from external peripheral devices through the I/O bus.

4.2.1 Overall Operation

The overall organization of the DATA 620/i I/O system, including a typical set of peripheral devices, is shown in figure 4-1. Standard or special peripheral devices are in parallel on the I/O bus.

Two types of I/O operations may be performed: program control and automatic control. Program-controlled information transfers between the central processor and the external devices to be executed are:

a. External control. An external control code may be transmitted, under program control, from the central processor to an external device.

b. Program sense. The central processor can sense the status of a selected external line under program control.

c. Single word transfer to/from A and B Registers. A single word may be transferred to or from the A and B registers under program control.

d. Single-word transfer to/from memory. A single word may be transferred to or from any memory location under program control.

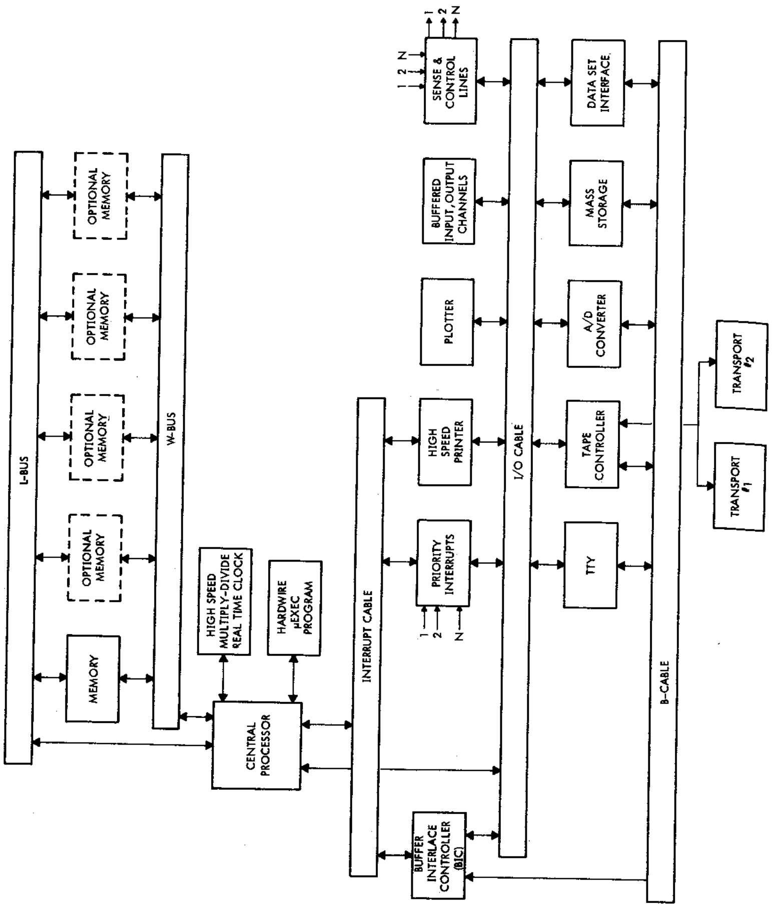


Figure 4-1. DATA 620/i System Organization.

The following types of automatically controlled information transfers between the central processor and the external devices may be executed independently of the program:

a. Program interrupt. An external device may force the program to execute an instruction at a specified location in memory.

b. Buffer interlace controller transfer to/from memory. Blocks of words may be transferred to or from sequential memory locations under control of an optional buffer interlace controller (BIC). Devices controlled by the BIC may also be operated under program control (single-word transfers).

c. Interlace data transfers. Single words may be transferred to or from memory by a special interface controller which uses the control signals available on the I/O bus.

4.2.2 Input/Output (I/O) Bus Structure

A typical organization of peripheral devices on the I/O bus is shown in figure 4-1. The complete I/O bus consists of two cables, the I/O cable and the interrupt cable. The I/O cable consists of the E bus, plus a set of control lines. The E bus contains 16 or 18 pairs of bidirectional lines which transmit control codes, addresses, and data between the central processor and the peripheral devices connected in parallel to this bus.

Information transfers are synchronized by peripheral controllers; these controllers may, in turn, control one or more peripheral devices. The central processor communicates directly with all peripheral controllers under program control. It may determine when a device is ready to send or receive information by sensing associated sense lines, or it may be notified by means of a program interrupt. Standard priority interrupt and sense line controllers are available, or special controllers may be provided. The interrupt cable is provided only for devices which use the program interrupt facility or the program trap facility.

Where block transfers of data, independent of, and concurrent with, internal operations are required (such as from tapes, drums, commutators, etc.) the buffer interlace controller may be provided. This element contains hardware registers which automatically generate the proper memory addresses for successive data transfers to or from the central processor memory, directly to or from the device through its controller.

This type of operation uses the program trap facilities of the computer. The trap sequence temporarily halts the program, without altering the program sequence, while the trapped I/O transfer occurs. Special interface designs may also take advantage of the trap facilities to control I/O transfers.

4.2.3 Input/Output Operations

During information transfers over the I/O bus, the E lines may carry control codes, addresses or data, depending upon which type of operation is being performed. Table 4-1 defines the I/O cable control signals used to synchronize all input/output operations. Table 4-2 summarizes the signals on the interrupt cable. Table 4-3 summarizes the signals present on the E bus during the program controlled I/O operations. Note that the I/O command is not transmitted intact over the E-bus. Bits 11-15 are decoded internally and only one of these lines will be true for each type of command. Bits 0-8 of the command are transmitted unchanged on the cable.

Table 4-1
I/O CABLE CONTROL LINE SIGNALS

CONTROL LINE	SIGNAL NAME	FUNCTION
Function Ready	FRYX-1	Indicates that the E-bus contains control or address information.
Data Ready	DRYX-1	Indicates that the E-bus contains data.
Sense Response	SERX-1	Indicates logical state of line queried by sense line address on E-bus.
Interrupt Acknowledge	IUAX-1	Indicates that external interrupt demand is being acknowledged. Address is placed on E-bus and removed when IUAX-1 goes false.
System Reset	SYRT-1	Reset line for initializing peripheral controllers. Energized by console RESET switch.

Table 4-2
INTERRUPT CABLE CONTROL LINE SIGNALS

CONTROL LINE	SIGNAL NAME	FUNCTION
Interrupt Request	IURX-1	Indicates a demand from the Interrupt module to force program to take one instruction from location specified by address on E-bus. This address will be placed on E-bus when IUAX-1 is true.
Trap Output Request	TPOX-1	Indicates that a buffer interlace controller or other trap device is requesting data transfer from memory.
Trap In Request	TPIX-1	Indicates that a buffer interlace controller or other trap device is requesting data transfer to memory.
Interrupt Clock	IUXC-1	1.1-MHz clock provided on cable for interrupt module. May be used in any interface design.
Priority Out	PRIX-1	Priority line used with interrupt and buffer interlace controller modules for priority determination.
Priority In	PR4X-1	Priority line returned to computer for permitting console interrupt.
Priority 2 and 3	PR2X-1, PR3X-1	Intermediate priority lines that are used to assign priority positions among trap and interrupt devices.
Interrupt Jump	IUJP-1	Indicates that instruction at interrupt location is a jump (2 word) instruction.

4.2.4 I/O Cable Adapter Card

The I/O cable adapter is a standard Micro-Versa LOGIC module IO-701 designed to facilitate interfacing with the DATA 620/i I/O bus. Typical examples illustrating its use are given in the interface reference manual. This card simplifies the use of many types of I/O interfaces.

4.3 PROGRAM CONTROL FUNCTIONS

Interfacing functions fall into two major categories: programmed operations, and automatic operations. The programmed operations are: External control (single-bit out), sense operations (testing a single bit), data transfer in (full-word inputs) and data transfers out (full-word outputs). The following paragraphs describe the programmed operations and examples of their use. The I/O instruction group is summarized in table G-11, appendix G.

4.3.1 External Control

The external control instruction is a single word, non-addressing instruction. It places a function code, contained in bits 0-8, on the E bus to effect a control operation on an external device.

EXC External Control Timing: 1 cycle

17	16	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
		10	0	XYY

(bits 17–16: 18-bit option)

The nine bits represented by XYY are placed on the E bus for transmission to the I/O controllers. The device address is contained in the YY portion of the data, and the X portion of the data contains the function to be performed by the selected device.

Indexing: No
Indirect Addressing: No
Registers Altered: None

4.3.2 Program Sense

The sense instruction is a double-word, addressing instruction which senses the logical state of an external line. Figure 4-2 shows the execution of this instruction.

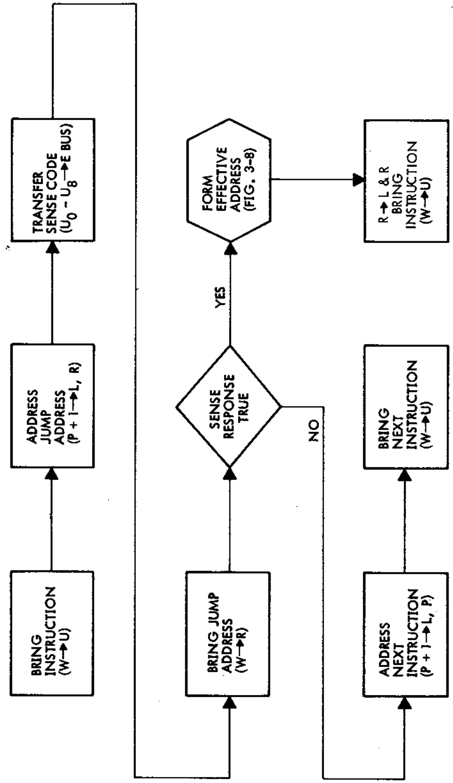


Figure 4-2. Sense Instruction, General Flow.

4.2.4 I/O Cable Adapter Card

The I/O cable adapter is a standard Micro-Versa LOGIC module IO-701 designed to facilitate interfacing with the DATA 620/i I/O bus. Typical examples illustrating its use are given in the interface reference manual. This card simplifies the use of many types of I/O interfaces.

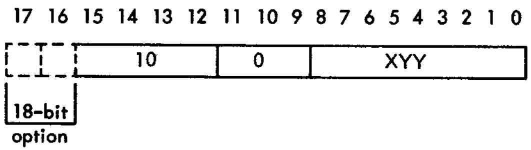
4.3 PROGRAM CONTROL FUNCTIONS

Interfacing functions fall into two major categories: programmed operations, and automatic operations. The programmed operations are: External control (single-bit out), sense operations (testing a single bit), data transfer in (full-word inputs) and data transfers out (full-word outputs). The following paragraphs describe the programmed operations and examples of their use. The I/O instruction group is summarized in table G-11, appendix G.

4.3.1 External Control

The external control instruction is a single word, non-addressing instruction. It places a function code, contained in bits 0-8, on the E bus to effect a control operation on an external device.

EXC External Control Timing: 1 cycle



The nine bits represented by XYY are placed on the E bus for transmission to the I/O controllers. The device address is contained in the YY portion of the data, and the X portion of the data contains the function to be performed by the selected device.

Indexing: No
Indirect Addressing: No
Registers Altered: None

4.3.2 Program Sense

The sense instruction is a double-word, addressing instruction which senses the logical state of an external line. Figure 4-2 shows the execution of this instruction.

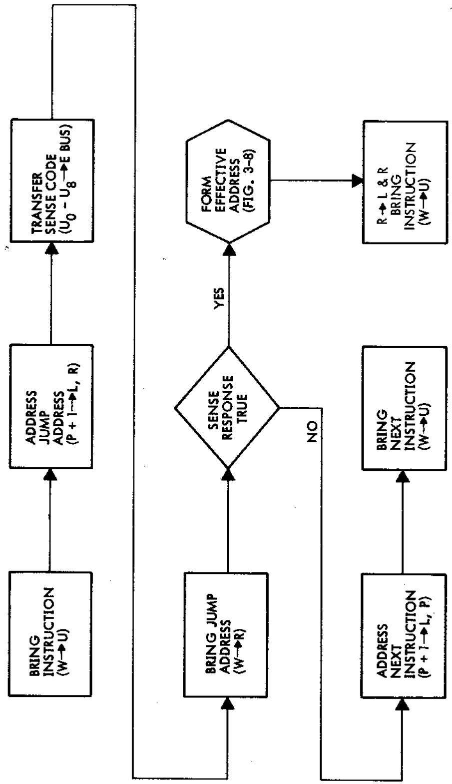


Figure 4-2. Sense Instruction, General Flow.

SEN Program Sense Timing: 2.25 cycles

	17 16 (18-bit option)	15	14 13 12 11	10 9	8 7 6 5 4 3 2 1 0
n		10		1	XYY
n+1		I	JUMP ADDRESS		

I = 0, word contains an address
I = 1, word contains an indirect address

The nine bits represented by XYY are placed in the party line I/O bus and represent the condition to be tested. X defines a specific line within device YY. The associated peripheral controller replies with either a true or false condition.

If a true condition is received by the DATA 620/i, a jump is made to the jump address. If a false condition is received the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

4.3.3 Data Transfer In

Two types of data transfer in instructions are provided: input to operational registers, and input directly to memory. The first type of input instruction is a single-word, non-addressing class instruction; the second type of input instruction is a double-word, addressing class instruction.

CIA Clear and Input to A Register Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	10	2	5ZZ

The A register is cleared and a data word from the selected device, ZZ, is transferred into the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

CIB Clear and Input to B Register Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	10	2	6ZZ

The B register is cleared and a data word from the selected device, ZZ, is transferred to the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

INA Input to A Register Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	10	2	1ZZ

A data word from the selected device, ZZ, is inclusively-ORed with the contents of the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

INB Input to B Register Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	10	2	2ZZ

A data word from the selected device, ZZ, is inclusively-ORed with the contents of the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

IME Input to Memory Timing: 3 cycles

	17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
n		10	2	0ZZ
n+1		Data Address		

A data word from the selected device, ZZ, is placed in the cleared effective memory address. Figure 4-3 shows the execution of this instruction.

Indexing: No
Indirect Addressing: No
Registers Altered: Memory

4.3.4 Data Transfer Out

Two types of data transfer out instructions are provided: output from operational registers, and output from memory. The first type of output instruction is a single-word, non-addressing class instruction; the second type is a double-word, addressing class instruction.

ØAR Output from A Register Timing: 2 cycles

17 16 (18-bit option)	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
	10	3	1ZZ

The contents of the A register are transferred to the selected device, ZZ.

Indexing: No
Indirect Addressing: No
Registers Altered: None

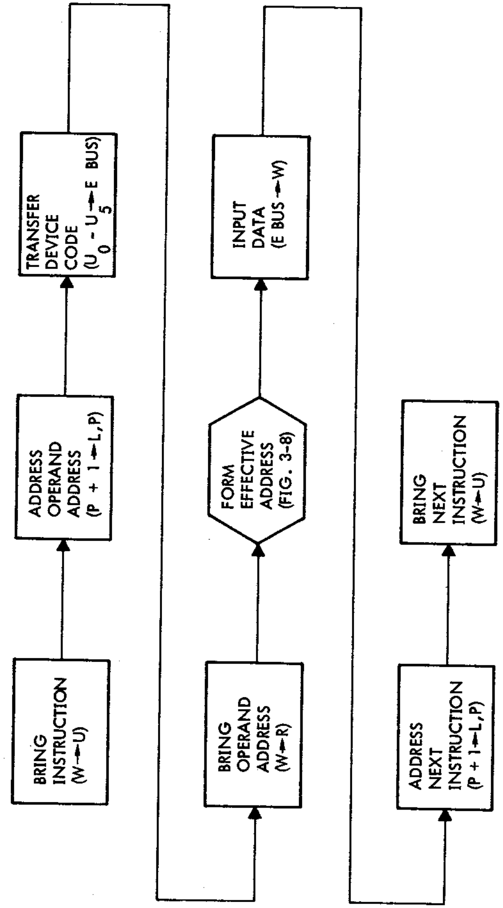
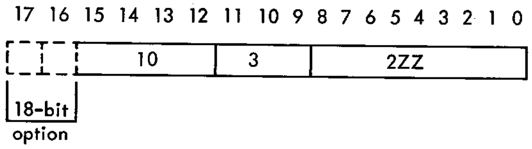


Figure 4-3. Input to Memory, General Flow.

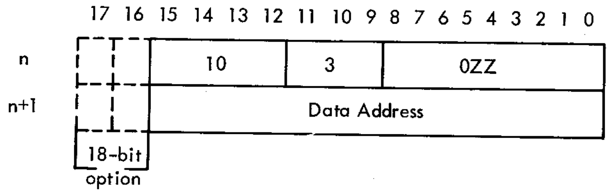
ØBR Output from B Register Timing: 2 cycles



The contents of the B register are transferred to the selected device, ZZ.

Indexing: No
Indirect Addressing: No
Registers Altered: None

ØME Output from Memory Timing: 3 cycles



The contents of the effective memory address are transferred to the selected device, ZZ.

Indexing: No
Indirect Addressing: No
Registers Altered: None

4.4 AUTOMATIC CONTROL FUNCTIONS (optional)

Two types of computer timing sequences are provided to automatically transfer control and information signals between the I/O and the DATA 620/i:

a. An interrupt timing sequence is initiated when the DATA 620/i recognizes an external interrupt signal. This sequence forces the computer to execute an instruction at the memory location specified by interrupt logic through the E bus.

b. A trap timing sequence is initiated when an external device signals that it wishes to transfer a word to or from memory. The external device must supply the memory address of the word through the E bus. This sequence delays the internal program sequence for the time required to execute the I/O transfer (2.7 μsec).

The devices that demand either of these automatic sequences must first establish a priority to resolve two or more simultaneous demands for service. The priorities of devices demanding service are determined every 1.8 μsec, and are clocked by the interrupt clock (refer to table 4-1).

The basic computer has one built-in priority device, the power failure interrupt. The power failure interrupt is permanently wired for the highest priority. Unless power failure (scanned every 1.8 μsec) is detected, the computer will service interrupt or trap requests from the interrupt cable on a priority basis.

Priority assignment for devices on the I/O cable is optional and is a part of the system definition. Priorities may be fixed for any given configuration by properly connecting priority lines in the I/O cable. Priorities can be altered if the definition changes.

4.4.1 Program Interrupt (optional)

The DATA 620/i has a multi-level interrupt system with single-execute, on/off and selective arm/disarm capability. Each interrupt line is assigned a unique memory destination address which is the first of a pair of locations. The system is modular and expandable in sets of eight levels.

Each optional interrupt line has an enable/disable flip-flop which is addressable and set by interrupt control instructions. If signals exist on one or more interrupt lines, the highest-priority line is recognized and the corresponding memory destination address is transmitted to the DATA 620/i after the current instruction is executed.

The program can maintain the hardware order of priority levels, or a re-order to meet dynamic queuing. For each group the order is determined by an 8-bit mask word transferred by the program to the arm/disarm flip-flops in the interrupt system. The action initiated by the interrupt subroutine causes the interrupting device to remove its requesting signal.

An acknowledgement of an interrupt causes the instruction located at the destination address to be executed. The instruction can be any of the DATA 620/i repertoire. This technique permits the interrupts to be of the "single execute" type, whereby single-instruction responses to external signals can be serviced in one instruction period. A real-time clock can be implemented with an interrupt line and an external pulse generator. An automatic data channel can be implemented with as few as two interrupt lines. If the executed instruction is a jump, the interrupt system is automatically inhibited permitting the inhibit to be terminated under program control. While in the inhibit mode, the interrupt subroutine may selectively enable and disable levels, and then enable the system permitting the selected levels to interrupt the level being processed.

4.4.2 Interlace Data Transfers (optional)

Interlace data transfers may be performed concurrently with internal program operation. This type of operation uses the computer trap timing sequence to delay the program for 2.7 μsec while a word is transferred between memory and a peripheral device. The transfer is controlled by the external device which must transmit the memory address of the data word, and must synchronize the operation using the signals transmitted over the I/O control lines (table 4-1). The maximum interlace transfer rate is 202,000 words per second.

The general trap sequence flow is shown in figure 4-4. The maximum computer delay in acknowledging a trap request is 5.4 μsec. However, the time delay experienced by a specific controller in receiving acknowledgement to a trap request may be extended by the time required for the central processor to service higher-priority requests.

Special peripheral controllers designed for system applications (such as A/D and D/A converters, etc.) may utilize the trap facilities of the computer to implement automatic I/O operations (refer to the interlace reference manual for detailed design information). A standard buffer interlace controller is also available for use with all standard DATA 620/i peripheral equipment. Special system devices may also be interfaced for interlace operations under control of this unit.

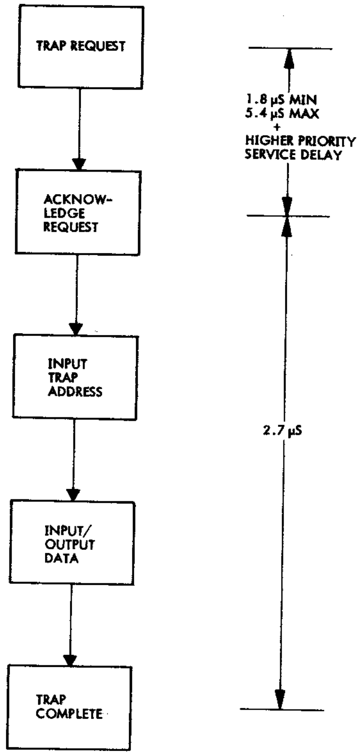


Figure 4-4. Trap Sequence, General Flow.

SECTION V
CONTROL CONSOLE OPERATION

5.1 CONTROLS AND INDICATORS

The DATA 620/i console (figure 5-1) provides controls and displays required for operator communication with the computer. Console facilities are of two kinds: register display and control switches. The contents of all operational registers including the instruction register, can be displayed in binary-octal form. During normal operation (run mode) the contents of the computer C-bus are displayed continuously. Data entry into a selected operational register is accomplished in step mode (computer halted) by mementary contact lever action switches. During run mode, these switches are deactivated to prevent accidental alteration of the register contents.

Control switches allow the operator to manually alter normal program operation. These switches described in table 5-1, provide considerable control flexibility, and are useful for maintenance, troubleshooting, and program debugging. The sense switch controls are also useful in normal program operation to allow selection by the operator of particular program sequences to be executed.

Table 5-1
CONTROLS AND INDICATORS

CONTROL OR INDICATOR	FUNCTION
Register Display	In-line display of 16 (or 18) bits in selected operational register. Register bits are numbered from right to left with the sign bit appearing on the far left side of the display. Lights are grouped in an octal arrangement. Selection of the register to be displayed is accomplished by the register select switches.
Register Select Switches	Five alternate action switches used to select one of five registers for display. Only one register may be selected at a time. Selection of two or more at the same time disables the selection logic and the display becomes blank.

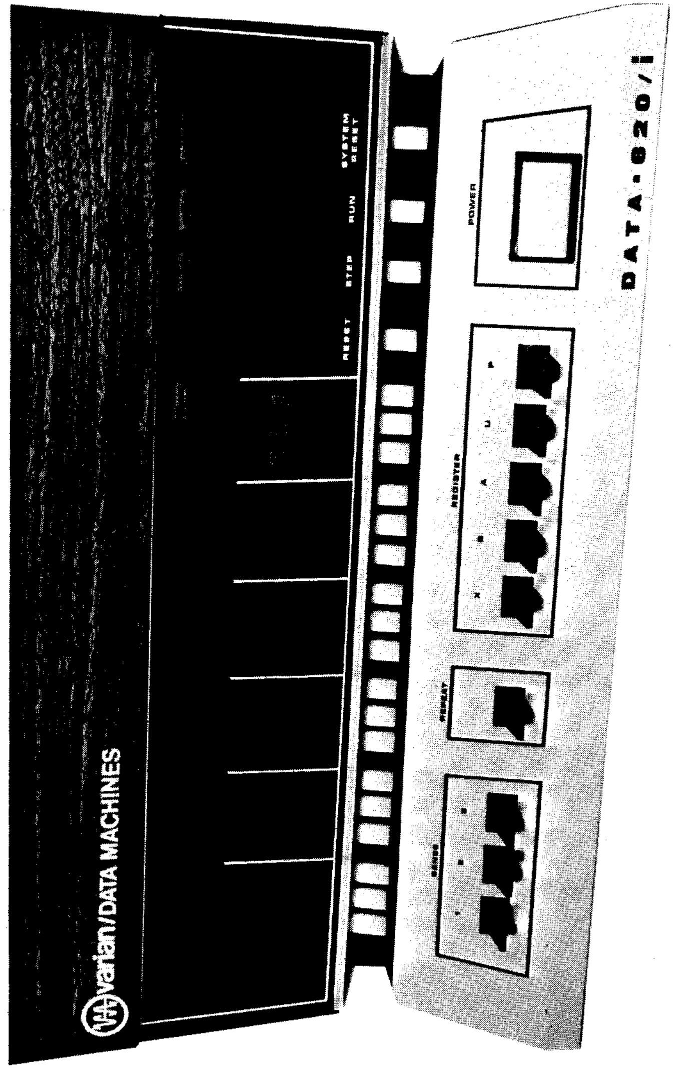


Figure 5-1. Control Console.

Table 5-1 (continued)
CONTROLS AND INDICATORS

CONTROL OR INDICATOR	FUNCTION
Status Display	Four indicators are provided to indicate the status of the machine. Overflow status indicator lights when the overflow flip-flop is on. STEP indicator lights when the computer is in step mode and μexec facility is not being used. RUN indicator lights when the computer is in run mode. ALARM is an indicator used to flag a thermal overload condition. It also lights when power is applied to the computer through the system circuit breaker but power ON/OFF switch on the console is in the OFF position.
RESET Switch	The RESET switch causes the selected register to be cleared. This switch is disabled when the computer is in the run mode.
STEP Switch	The STEP switch is a momentary contact switch that causes the instruction in the instruction register to be executed if the computer is in the step mode. If the computer is in the run mode, pressing the STEP switch causes the computer to halt at the completion of the instruction being executed.
RUN Switch	The RUN switch causes the program to run at the location specified by the program counter after first executing the instruction in the instruction register.
SYSTEM RESET	The SYSTEM RESET switch is a system clear control that forces the computer to halt mode, and initializes control flip-flops in the processor. In addition, all peripheral devices are initialized by SYSTEM RESET. The control is normally used as an initialize control, but is useful to halt I/O operations.

Table 5-1 (continued)
CONTROLS AND INDICATORS

CONTROL OR INDICATOR	FUNCTION
REPEAT	Alternate-action switch that permits manual repeat of an instruction in instruction register. Pressing STEP switch executes instruction and advances program counter; however, contents of the instruction register are left unchanged. Switch on the control console is activated only when the STEP light is on (operation halted).
SENSE Switches 1, 2, 3	Alternate-action switches that permit manual program control whenever the sense switch jump, or jump-and-mark, or execute instructions (JSS1, JSS2, JSS3, JS1M, JS2M, XS1, XS2, XS3) are performed. The indicated jump and execute operations are performed only if the corresponding sense switch is ON.
POWER ON/OFF	Alternate-action switch/indicator turns power supplies on and off. Indicator/switch is illuminated when power on; indicator is off when power is off.

5.2.2 Manual Program Entry and Execution

When the computer is halted (step mode), programs and data may be read from memory and entered into memory, and a pre-stored program may be manually executed.

To load words into memory (either instructions or data), set up the desired word in the A, B, or X register. Set up the appropriate store-type instruction (STA, STB, STX) with the desired operand address in the instruction (U) register and press the STEP switch to execute the store operation.

To display the contents of any memory cell in the A, B, or X register; set up the appropriate load-type instruction (LDA, LDB, LDX) with the proper memory address in the instruction register; then press the STEP switch to load the selected word into the register.

To manually execute a program stored in memory, set up the starting location of the program in the program counter. When the STEP switch is pressed, the instruction contained in the instruction register is executed, and the instruction of the selected location is transferred to the instruction register. Repeated operation of the STEP switch will then step through the program one instruction at a time. All operations such as multi-level indirect addressing will be performed for each instruction each time the STEP switch is operated. Note that I/O instructions that involve an asynchronous device which transfers data in a block such as magnetic tape or the teletype generally cannot be operated in a single-step mode.

5.2.3 Instruction Repeat

In the step mode, the instruction register contains the next instruction to be executed when STEP is pressed. The program counter contains the location of the next instruction to be transferred to the instruction register after the current instruction is executed.

In some cases, it is desirable to manually execute an instruction several times. When the REPEAT switch is on, instruction register loading (when STEP is pressed) is inhibited even though the instruction counter is advanced each time. This mode is particularly useful for loading words into sequential memory locations, or for displaying the contents of sequential memory locations, or for displaying the contents of sequential memory cells.

To load a group of sequential memory cells, set up the appropriate store-type instruction (STA, STB, STX) in the instruction register with the relative address mode in the m field and the base address in the a field. Repeated operation of the STEP switch will store the contents of A, B, or X into sequential memory locations. The word loaded on each step may be changed by entering the desired value into the operational register for each step.

To display the contents of a group of sequential memory cells, set up the appropriate load-type instruction (LDA, LDB, LDX) in the instruction register, in the relative address mode, with the base address in the instruction register and the a field = 0. The contents of the sequential locations will be displayed in the selected operational register with each operation of the STEP switch.

5.2.4 Sense Switches

The SENSE switches allow the operator to dynamically alter a program sequence in either the run or step mode. The three SENSE switches provide a logical-AND function with bits 6-8 of the instruction word, and consequently can be used for various logical branches set up on the console.

Table 5-1 (continued)
CONTROLS AND INDICATORS

CONTROL OR INDICATOR	FUNCTION
REPEAT	Alternate-action switch that permits manual repeat of an instruction in instruction register. Pressing STEP switch executes instruction and advances program counter; however, contents of the instruction register are left unchanged. Switch on the control console is activated only when the STEP light is on (operation halted).
SENSE Switches 1, 2, 3	Alternate-action switches that permit manual program control whenever the sense switch jump, or jump-and-mark, or execute instructions (JSS1, JSS2, JSS3, JS1M, JS2M, XS1, XS2, XS3) are performed. The indicated jump and execute operations are performed only if the corresponding sense switch is ON.
POWER ON/OFF	Alternate-action switch/indicator turns power supplies on and off. Indicator/switch is illuminated when power on; indicator is off when power is off.

5.2.2 Manual Program Entry and Execution

When the computer is halted (step mode), programs and data may be read from memory and entered into memory, and a pre-stored program may be manually executed.

To load words into memory (either instructions or data), set up the desired word in the A, B, or X register. Set up the appropriate store-type instruction (STA, STB, STX) with the desired operand address in the instruction (U) register and press the STEP switch to execute the store operation.

To display the contents of any memory cell in the A, B, or X register; set up the appropriate load-type instruction (LDA, LDB, LDX) with the proper memory address in the instruction register; then press the STEP switch to load the selected word into the register.

To manually execute a program stored in memory, set up the starting location of the program in the program counter. When the STEP switch is pressed, the instruction contained in the instruction register is executed, and the instruction of the selected location is transferred to the instruction register. Repeated operation of the STEP switch will then step through the program one instruction at a time. All operations such as multi-level indirect addressing will be performed for each instruction each time the STEP switch is operated. Note that I/O instructions that involve an asynchronous device which transfers data in a block such as magnetic tape or the teletype generally cannot be operated in a single-step mode.

5.2.3 Instruction Repeat

In the step mode, the instruction register contains the next instruction to be executed when STEP is pressed. The program counter contains the location of the next instruction to be transferred to the instruction register after the current instruction is executed.

In some cases, it is desirable to manually execute an instruction several times. When the REPEAT switch is on, instruction register loading (when STEP is pressed) is inhibited even though the instruction counter is advanced each time. This mode is particularly useful for loading words into sequential memory locations, or for displaying the contents of sequential memory locations, or for displaying the contents of sequential memory cells.

To load a group of sequential memory cells, set up the appropriate store-type instruction (STA, STB, STX) in the instruction register with the relative address mode in the m field and the base address in the a field. Repeated operation of the STEP switch will store the contents of A, B, or X into sequential memory locations. The word loaded on each step may be changed by entering the desired value into the operational register for each step.

To display the contents of a group of sequential memory cells, set up the appropriate load-type instruction (LDA, LDB, LDX) in the instruction register, in the relative address mode, with the base address in the instruction register and the a field = 0. The contents of the sequential locations will be displayed in the selected operational register with each operation of the STEP switch.

5.2.4 Sense Switches

The SENSE switches allow the operator to dynamically alter a program sequence in either the run or step mode. The three SENSE switches provide a logical-AND function with bits 6-8 of the instruction word, and consequently can be used for various logical branches set up on the console.

PROGRAMMING REFERENCE

SECTION I
GENERAL DESCRIPTION

1.1 INTRODUCTION

The DATA 620/i computer is a high-speed, parallel binary computer. Its extensive instruction repertoire, flexible input/output system, and modular packaging make the DATA 620/i computer ideally suited for operation as a general-purpose computer or as a system component. The computer, simple in design, is easy to program, operate, and maintain. As a system component, the computer is easily integrated with other equipments through the use of standard or special peripheral interface elements.

Features of the DATA 620/i computer are:

- Fast Operation	1.8-microsecond memory cycle.
- Large Instruction Repertoire	107 standard instructions with over 128 micro-instructions and 18 optional instructions.
- Expandable Word Length	16 or 18-bit word arithmetic.
- Modular Memory	4096 words minimum, 32768 words maximum.
- Multiple Addressing Modes	Five types: direct, indirect, relative, index, immediate, and extended (optional).
- Flexible I/O System	64 device addresses on the standard I/O bus; optional, fully-buffered input/output and direct memory access are available.
- Extensive Software	Programming and diagnostic aids such as assembier and procedure-oriented programs required for efficient computer use.
- Modular Packaging	Mounts in a standard 19-inch cabinet. No special mechanical or environmental facilities are required.

1.2 PURPOSE OF THE MANUAL

This manual provides the DATA 620/i computer programmer with the information necessary to use the DATA 620/i assembly system, the utility and program diagnostic package (AID), the symbolic correction program (COR), and the symbolic tape source correction program (EDITOR). Before this manual can be used effectively, the programmer should be familiar with the contents of the DATA 620/i system reference manual, which contains a detailed description of the DATA 620/i computer. Table 1-1 lists all manuals pertaining to the DATA 620/i computer and peripheral controllers.

1.3 COMPUTER ORGANIZATION

The DATA 620/i is organized with a unique bus structure, selection logic, and eight registers. The organization provides universal information routing, buffered processing, micro-programming capability, indexing without time penalty, and buffered input/output data transfer. A unique optional facility, Micro-EXEC, is also available which permits complex algorithms to be implemented with external control hardware. This capability provides increases in processing speed in excess of 400 percent over normal programmed operations.

The organization of the DATA 620/i is shown in figure 1-1. This diagram shows the major functional elements of the machine, including the registers and busses provided for information transfer.

The major functional elements of the DATA 620/i, indicated in figure 1-1, are: memory, control section, arithmetic/logic section, operational registers, internal busses, and input/output (I/O) bus.

1.3.1 Memory

The internal storage of the computer consists of 4096-word modules connected to the L and W busses. The mainframe can accommodate one 4096-word module. Additional modules are added in an additional frame that is attached to the mainframe. The computer memory can be expanded to a maximum of 32,768 words using 4096-word modules.

Instruction words read from memory are transferred to the control section for execution. Words may be transferred, under program control, from memory to the arithmetic/logic section, to the operational registers, or to the I/O bus. Words may be transferred, under program control, to memory from the operational registers or the I/O bus.

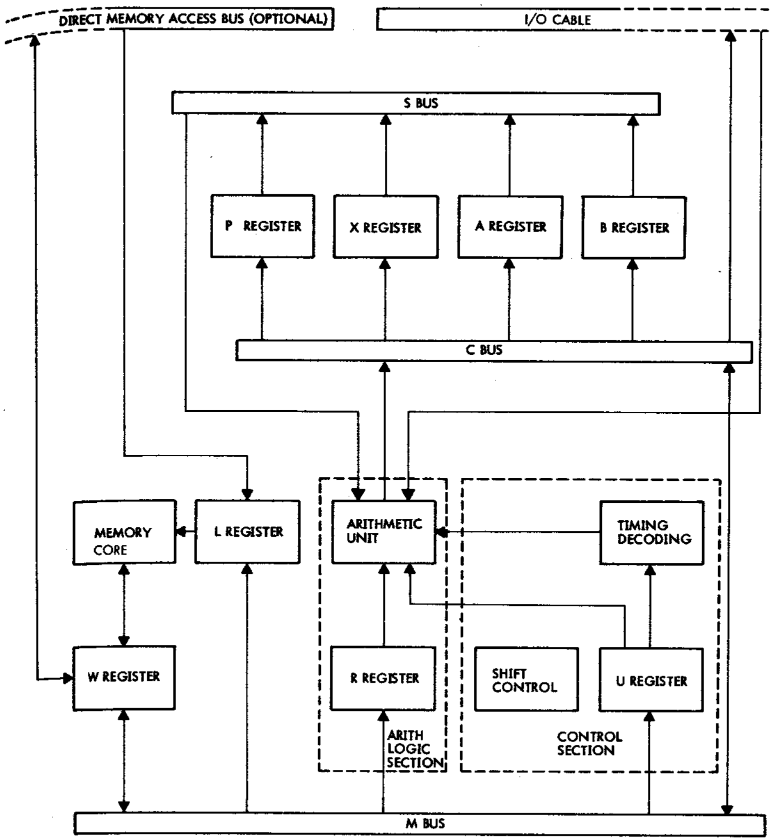


Figure 1-1. DATA 620/i Organization.

Table 1-1
DATA 620/i DOCUMENTS

PUBLICATION NUMBER	TITLE
VDM 3000	System Reference Manual
VDM 3001	Interface Reference Manual
VDM 3002	Programming Reference Manual
VDM 3003	FORTRAN Manual
VDM 3004	Subroutine Manual
VDM 3005	Maintenance Manuals
VDM 3006	ASR-33 Teletype Controller Reference Manual
VDM 3007	Buffer Interlace Controller Reference Manual
VDM 3008	Magnetic Tape Controller Reference Manual
VDM 3009	600 LPM Line Printer Controller Reference Manual
VDM 3010	300 LPM Line Printer Controller Reference Manual
VDM 3011	Paper Tape System Controller Reference Manual
VDM 3012	100 CPM Card Reader Controller Reference Manual
VDM 3013	Priority Interrupt Reference Manual
VDM 3014	A/D Converter Reference Manual
VDM 3015	Optical Scanner Controller Manual
VDM 3016	ASR-35 Teletype Controller Reference Manual
VDM 3017	Digital Plotter Controller Reference
VDM 3018	DDC Disc Controller Reference Manual
VDM 3019	Console Printer Controller Reference Manual

When one or more optional buffer interlace controller (BIC) is used, the system is capable of direct transfer between memory and peripheral devices on the I/O bus, concurrent with computations.

1.3.2 Control Section

The control section provides the timing and control signals required to perform all operations in the computer. The major elements in the section are the U register, the timing and decoding logic, and the shift control.

The U register (instruction register) is 16 bits long. This register receives each instruction from memory through the W bus and holds the instruction during its execution. The control fields of the instruction word are routed to the decoding and timing logic where the codes determine the required timing and control signals. The address field from U, used for various addressing operations, is also routed to the arithmetic/logic section.

The decoding logic decodes the fields of the instruction word held in U to determine the control signal levels required to perform the operations specified by the instruction. These levels select the timing signals generated by the timing unit.

Timing logic generates the basic 2.2-MHz system clock. From this clock, timing logic derives the timing pulses which control the sequence of all operations in the computer.

The shift control contains the shift counter and logic which control operations performed by the shift, multiply, and divide instructions.

1.3.3 Arithmetic/Logic Section

This section consists of two elements; the R register and the arithmetic unit.

The R register receives operands from memory and holds them during instruction execution. The operand may be either data or address words. This register permits transfers between memory and I/O bus during the execution of extended-cycle instructions.

The arithmetic unit contains gating required for all arithmetic, logic, and shifting operations performed by the computer. Indexed and relative address modifications are performed in this section without increased instruction execution time.

The arithmetic unit also controls the gating of words from the operational registers and the I/O bus onto the C bus where they are distributed to the operational registers or to memory registers. This facility is used to implement many of the microinstructions of the computer.

1.3.4 Operational Registers

The basic DATA 620/i computer contains eight registers.

The operational registers consist of the A, B, X, and P registers. The A, B and X registers are directly accessible to the programmer. The P register is indirectly accessible through use of the jump class instructions which modify the program sequence. The operational registers are described in the following paragraphs.

A register. This full-length, 16/18-bit register is the upper half of the accumulator. This register accumulates the results of logical and addition/subtraction operations, the most-significant half of the double-length product in multiplication, and the remainder in division. It may also be used for input/output transfers under program control.

B register. This full-length, 16/18-bit register is the lower half of the accumulator. This register accumulates the least-significant half of the double-length product in multiplication, and the quotient in division. It may also be used for input/output transfers under program control and as a second hardware index register.

X register. This full-length 16/18-bit register permits indexing of operand addresses without adding time to execution of indexed instructions.

P register. This full-length, 16/18-bit register holds the address of the current instruction and is incremented before each new instruction is fetched. A full complement of instructions is available for conditional and unconditional modification of this register.

S register. This five-bit register controls the length of shift instructions in combination with the U register. This register also buffers memory from the control unit.

1.3.5 Internal Busses

C bus. This bus provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O bus, the operational registers, and the memory registers. The console display indicators are also driven from the C bus. Distribution of data simultaneously to multiple operational registers is facilitated by this bus.

S bus. This bus provides the parallel path and selection logic for routing data from the operational registers to the arithmetic unit.

W bus. The memory word (W) register is directly connected to all memory modules through the W bus. The bus is bydirectional and time-shared among memory modules.

L bus. The memory address (L) register is directly connected to all memory modules through the L bus. The bus is unidirectional.

1.3.6 Input/Output (I/O) Bus

The bidirectional I/O bus provides the parallel path between the computer and all peripheral devices. This bus contains the data and control lines required for transmitting ready, sense, function, and interrupt signals as well as data words between the computer and peripheral devices.

1.3.7 Direct Memory Access (DMA)

The DMA option allows data transfer into or out of memory modules without disturbing the contents of the operational registers. Only the L and W registers are altered. Access to memory using the DMA facility is on a "cycle-steal" basis and requires 2.7 microseconds of processor time per transfer.

SECTION II

DATA 620/i ASSEMBLY SYSTEM

2.1 INTRODUCTION

The DATA 620/i assembler (DAS) assists in program preparation by allowing instructions, addresses, address modifiers, and constants to be specified in a straightforward and meaningful manner. Instruction mnemonics such as STB (store B register) are used in place of numeric instruction codes. Various memory locations (addresses) may be referred to by labels, not absolute locations. Constants may be entered into the DATA 620/i without converting the numbers into binary or octal form. Useful comments may be added either between symbolic statements or on the symbolic statement itself, to allow easy program check-out and documentation.

DAS reduces much of the tedious bookkeeping associated with machine language programming, but does not compromise the programmer's ability to fully utilize the DATA 620/i.

The basic assembly (DAS I) operates in a DATA 620/i system, which consists, as a minimum, of 4096 words of memory and an on-line teletype. The standard assembly (DAS I-F) requires 8192 words of memory.

Provisions have been made to utilize additional facilities such as magnetic tape, card reader, card punch, additional memory, and line printer if these components are available.

DAS is a two- or three-pass assembly system, which means that the source program must be read two or three times for complete assembly. During the first pass, values are assigned to all labels appearing in the location field (paragraph 2.2.3) and placed in the label table. During the second pass, the appropriate values for the instruction field and the variable field (paragraphs 2.2.4 and 2.2.5) are assembled into the object instruction and, together with the remarks field, are listed on the printer. During pass three, the object instructions are punched onto paper tape. In certain peripheral I/O configurations, passes two and three are combined.

2.2 THE DAS SOURCE LANGUAGE

2.2.1 Introduction

DAS translates symbolically coded instructions (the source program) into binary computer instructions (the object program). Except for certain pseudo instructions (paragraph 2.4), each symbolic source statement will generate one computer instruction.

Computer instructions generated by DAS fall into two categories, instructions and data. The instructions are described in paragraph 2.3 and the data is described in paragraph 2.2.6.

A source statement consists of several parts, or fields. Each source statement may contain a combination of these fields depending on the requirements of the instruction or pseudo instruction being processed. The fields are: location, instruction, variable, and remarks fields.

2.2.2 DAS Characters

The following characters are recognized by the DAS assembler:

Alphabetic characters

ABCDEFGHIJKLMNOPQRSTUVWXYZ$

Numeric characters

0123456789

Special characters

+ (plus sign)	) (right parenthesis)	← (left arrow)*
- (minus sign)	b (blank)	\ (back slash)
* (asterisk)	@ (at sign)	! (exclamation point)
/ (slash)	] (left bracket)	" (quotes)
. (period)	[(right bracket)	# (pound sign)
= (equal sign)	< (less than sign)	% (percent sign)
, (comma)	> (greater than sign)	& (ampersand)
' (prime)	? (question mark)	: (colon)
((left parenthesis)	↑ (up arrow)	; (semi-colon)

*replaced by blank on magnetic tape.

Teletype characters

CR (carriage return)

LF (line feed)

The SYMBOLIC LISTING is formated as an 8-1/2 by 11 page with a one inch margin at top and bottom.

The OBJECT PROGRAMS are prepared in standard binary format.

2.2.3 Location Field

Labels in the location field consist of from one to four alphanumeric characters, the first of which is alphabetic. Special characters are not allowed in a label. Additional alphanumeric characters may be added to the first four characters of the label to form an extended label for the convenience of the programmer. However, the assembler recognizes only the first four characters. Labels are usually attached to only those source statements that are referred to elsewhere in the program, but this is not a requirement. Values are attached to the labels during the first pass of the assembler.

2.2.4 Instruction Field

The instruction field contains special operation code mnemonics which describe the computer instructions. The same mnemonic may be used both in the instruction field and in the location field without conflict. An asterisk (*) following the instruction mnemonic indicates indirect addressing.

Operation code mnemonics may be redefined by the pseudo instruction OPSY (paragraph 2.4.3).

2.2.5 Variable Field

The purpose of the variable field varies with the needs of the individual instruction. The variable field may consist of a label, a constant, or an expression which consists of a combination of labels and constants. The expressions that may be used in the DAS assembly system are similar to arithmetic expressions, except that no parentheses may be used. The following arithmetic operators are available in the variable field of DAS:

+ (addition)

- (subtraction)

* (multiplication)

/ (division)

All arithmetic operations are performed in the integer mode, i.e., modules 2^{15}. The expression A+B/C*D is equivalent to the algebraic expression A+(B/C)*D. The operations are performed from left to right with the multiply and divide operations taking precedence over the add and subtract operations.

Access to the current value of the location counter may be gained by the special element *, when used as the first character of the variable field. An asterisk immediately preceding an operator is treated as the location counter rather than an operator. Thus, the expression *+1 is interpreted as meaning the current value of the instruction counter plus one.

Constant-generation facilities available in the DAS assembly system are described in the following paragraphs.

2.2.5.1 Decimal Integers. A decimal integer is an optionally signed string of from one to six digits, the first of which is not zero.

Example: 1, 7, -3, +327

2.2.5.2 Octal Integers. An octal integer is an optionally signed string of from one to seven octal digits, the first of which is always zero

Example: 07, -044, +014

2.2.5.3 Floating-Point Numbers.

Floating-point numbers can be assembled by DAS in one of the following forms:

) + integer. fraction E ± exponent
) 375.64E+7
) 9.E-2, .1E+12
) -4.+20

A right parenthesis, digit, and decimal point must be present. All other items are optional.

Location field: blank
Instruction field: DATA
Variable field: One or more floating-point numbers separated by commas.

The format of the assembled data is shown below.

16-BIT FORMAT

	15	14 13 12 11 10 9 8 7 6	5 4 3 2 1 0
L	S	Exponent + 0200	High Mantissa
L+1	X	Low Mantissa	

18-BIT FORMAT

	17	16 15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
L	S	Exponent + 0200	High Mantissa
L+1	X	Low Mantissa	

The sign bit of the second word is always set to zero.

Negative data are in 1's complement form in the first word.

2.2.5.4 Alpha Constant. An alpha constant is a string of characters enclosed by primes ('). An alpha constant is represented internally as an 8-Bit ASCII Code. When one character is generated, the character is right-justified with leading zeros. Each memory location may contain two characters. A blank in the string is recognized as a character.

Some examples of words generated by character constants are given below:

	17 16	15 ... 8	7 ... 0	
'A'	0 0	0 0 0 0 0 0 0 0	1 1 0 0 0 0 0 1	One word is generated.
		0	A	
'AB'	0 0	1 1 0 0 0 0 0 1	1 1 0 0 0 0 1 0	One word is generated.
		A	B	
'ABC'	0 0	1 1 0 0 0 0 0 1	1 1 0 0 0 0 1 0	Two words are generated. Note that the space character code is used to fill the low-order eight bits of the second word if an odd number (except 1) of characters is specified within the primes.
		A	B	
	0 0	1 1 0 0 0 0 1 1	1 0 1 0 0 0 0 0	
		C	Space	

If the DATA 620/i has an 18-bit word length, zeros are generated in bits 16 and 17 of each word.

2.2.5.5 Address Constant. An address constant consists of a label, number or expression enclosed in parentheses, and generates a 15-bit address with bit position 15 set to a logical 0 to indicate a direct address.
Example: (A+2), (3), (A)
A is an address symbol and its value is obtained from the Label table. If the program is relocated, the value of the address constant is changed to agree with the location assigned to the instruction labeled A.

2.2.5.6 Indirect Address Constant. An indirect address constant consists of an address constant followed by an asterisk (*), and generate a 15-bit address with bit position 15 set to a logical 1 to indicate an indirect address.
Example: (A)*, (A+3)*, (3)*

2.2.5.7 Literals. Literals allow the programmer to refer to a constant in the variable field and have DAS generate the data and assign a location in memory. Even though a literal may be used many times, only one location will be generated.

A literal reference is indicated by an equal sign (=) followed by any format of a one-word constant (paragraph 2.2.6).

Examples: =3 =+3 =-3 =044 =(A+2)* ='A' ='GO'

For certain instructions, more than one expression is desired. In these cases the expressions are separated by commas (,).

Note that the expressions deal with the values assigned to labels, and not the contents of memory locations that may be referenced by the labels.

2.2.6 Remarks Field

The remarks field is separated from the variable field by at least one blank character. The information in the remarks field is ignored by the DAS assembler and the programmer may put in any comments that help him in documentation and debugging.

2.3 DATA 620/i INSTRUCTIONS

2.3.1 Introduction

The following paragraphs assume the 16-bit configuration of the DATA 620/i. Each of the four instruction types is described in the following paragraphs. Optional Instructions are recognized only when installed in the object computer.

2.3.2 Type-1 Instructions

Type-1 instructions occupy one computer word and are addressable. DAS recognizes the following forms:

LOCATION FIELD	INSTRUCTION FIELD	VARIABLE FIELD	COMMENTS
Label	Inst. Mnemonic	Expression	The expression value is the effective address.
Label (label is optional)	Inst. Mnemonic	Exp 1, Exp 2	The value (modulo 512) of expression Exp 1 is added to the contents of the X register or the B register to form the effective address. The expression Exp 2 must have a value of 1 or 2 to designate the X or the B register, respectively.
Label	Inst. Mnemonic*	Expression	The expression value is the indirect address of the operand.
Label	Inst. Mnemonic	(Expression)*	

If the first form of the instruction listed above is used, DAS will choose the addressing mode of the generated computer instruction according to the following rules:

a. If the specified address lies within core locations 0-2047 inclusively, the direct address will be used.

b. If the specified address lies outside core locations 0-2047 but not more than 512 and not less than one word beyond the current instruction, the mode of addressing is relative to the location counter.

c. If neither condition a nor condition b is true, a 15-bit address will be generated in memory area 0-511 (called bank 0), and the bank 0 address will be used in the instruction in the indirect mode.

Type 1 mnemonics recognized are:

LDA	(load A register)	INR	(increment memory word)
LDB	(load B register)	ERA	(exclusive-OR to A register)
LDX	(load X register)	ØRA	(inclusive-OR to A register)
STA	(store A register)	ANA	(AND to A register)
STB	(store B register)	MUL	(optional multiply)
STX	(store X register)	DIV	(optional divide)
ADD	(add to A register)		
SUB	(subtract from A register)		

2.3.3 Type-2 Instructions

Type-2 instructions require two computer words. The second word is the direct or indirect address if the instruction is a jump, jump-and-mark, or execute. The second word of an Immediate instruction is the operand. The second word of the byte or extended address instruction is the operand address. DAS recognizes the following forms:

LOCATION FIELD	INSTRUCTION FIELD	VARIABLE FIELD	COMMENTS
Label	Inst. Mnemonic	Expression	The expression value is the effective jump, jump-and-mark, or execute address, or it is the operand of an Immediate instruction.
Label	Inst. Mnemonic*	Expression	The expression value is the indirect jump, jump-and-mark, or execute address.
Label (label is optional)	Inst. Mnemonic	(Expression)*	

The following type-2 mnemonics are recognized as Immediate instructions:

LDAI	STAI	ADDI	ERAI	DIVI (optional)
LDBI	STBI	SUBI	ØRAI	MULI (optional)
LDXI	STXI	INRI	ANAI	

The following type-2 mnemonics are recognized as jump, jump-and-mark, and execute instructions:

JMP	JXZ	JANM	JS2M	XAZ
JØF	JSS1	JAPM	JS3M	XBZ
JAN	JSS2	JAZM	XEC	XXZ
JAP	JSS3	JBXM	XØF	XS1
JAZ	JMPM	JXZM	XAN	XS2
JBZ	JOFM	JS1M	XAP	XS3

The following type-2 mnemonics are recognized as byte instructions:

SLA	SSA	SLAC	SSAC	SCAE

2.3.4 Type-3 Instructions

Type-3 instructions are two-word computer instructions with a direct or indirect address in the second word. They differ from the type-2 instructions in that the variable field of the symbolic instruction contains two subfields instead of one.

DAS recognizes the following forms of type-3 instructions:

LOCATION FIELD	INSTRUCTION FIELD	VARIABLE FIELD
Label	Inst. Mnemonic	Exp 1, Exp 2
Label	Inst. Mnemonic*	Exp 1, Exp 2
Label	Inst. Mnemonic	Exp 1, (Exp 2)*
(label is optional)		

DAS recognizes the following type-3 mnemonics:

Dummy conditional jumps:
JIF (jump if...)
JMIF (jump-and-mark if...) or JIFM (jump-and-mark if...)
XIF (execute if...)

The value of the expression Exp 1 specifies which of the conditions will cause a jump, jump-and-mark, or execute instruction. The conditions of Exp 1 have the following values:

if OFLO set:	$0001_{(8)}$	if B = 0:	$0020_{(8)}$
if A < O:	$0004_{(8)}$	if X = O:	$0040_{(8)}$
if A ≥ O:	$0002_{(8)}$	if SS1 set:	$0100_{(8)}$
if A = O:	$0010_{(8)}$	if SS2 set:	$0200_{(8)}$

Compound conditions may be specified by adding together the values of the desired conditions.
For example:

INSTRUCTION FIELD	VARIABLE FIELD
JIF	0222, ALFA

Where 0220 = 0200 + 020 + 02 means: take the next instruction from address ALFA, if and only if, all three of the following conditions are true:

The A register contains a positive number:	0002
The B register contains zero:	0020
Sense switch 2 is set:	0200

The value of the expression Exp 2 is a direct or indirect jump, jump-and-mark, or execute address.

The following type-3 mnemonics are recognized as extended address instructions (optional):

LDAE	STAE	ADDE	ERAE	DIVE
LDBE	STBE	SUBE	ØRAE	MULE
LDXE	STXE	INRE	ANAE	

Type-3 instructions also include the following I/O instructions:

SEN	(sense for state of an I/O device)
IME	(input to memory)
ØME	(output from memory)

The value of the expression Exp 1 in the variable field of the instruction is the device subcode.

The value of the expression Exp 2 is a direct or indirect jump or memory address.

2.3.5 Type-4 Instructions

Type-4 instructions are one-word instructions which do not refer to a memory location. DAS recognizes the following formats:

LOCATION FIELD	INSTRUCTION FIELD	VARIABLE FIELD	COMMENTS
Label	Inst. Mnemonic		Variable field is blank.
Label	Inst. Mnemonic	Expression	The value of the expression specifies either source/ destination registers and overflow conditions, a shift count, an I/O device or function, or a halt number.
(label is optional)			

DAS recognizes the following type-4 mnemonics:

TZA, TZB, TZX	(clear register)
IAR, IBR, IXR	(increment register)
DAR, DBR, DXR	(decrement register)
CPA, CPB, CPX	(complement register)
TAB, TBA, TAX, TXA, TBX, TXB	(register transfer)
SØF, RØF	(overflow)
HLT, NØP	(control)

The following instruction mnemonics are functionally the same as the preceding register change instructions except that these mnemonics allow the user to specify multiple-source and/or destination registers, or specify whether or not function execution is dependent on the overflow conditions:

INSTRUCTION MNEUMONIC	INSTRUCTION FUNCTION
MERGE	Take the inclusive-OR of the contents of all specified source registers and deliver the result to each of the specified destination registers.
CØMPL	Like MERGE, except the result is ones-complemented before delivery.
INCR	Like MERGE, except + 1 is added to result before delivery.
DECR	Like MERGE, except + 1 is subtracted from the result before delivery.
ZERØ	Zero each of the specified destination registers.

The value of the expression used in the variable field of the instruction is interpreted by DAS as having the following meaning:

If bit 0 = 1 :	A is a destination register
If bit 1 = 1 :	B is a destination register
If bit 2 = 1 :	X is a destination register
If bit 3 = 1 :	A is a source register
If bit 4 = 1 :	B is a source register
If bit 5 = 1 :	X is a source register
If bit 8 = 1 :	The function is to be performed if and only if the overflow flip-flop is set to 1.

The instruction generated by DAS has the following format. Bits 8, 5, 4, 3, 2, 1, and 0 are extracted directly from the corresponding bits of the expression value.

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	0 V	f	f	X	B	A	X	B	A
18-bit Option									Conditional on Overflow	Function		Source Registers			Destination Registers		

The shift mnemonics recognized by DAS are listed below. The expression value represents the number of positions to be shifted. A value outside the range of 0-31 is reduced modulo 31 and an error code is printed.

LSRA	(logical shift right, A)
LRLA	(logical rotate left, A)
LSRB	(logical shift right, B)
LRLB	(logical rotate left, B)
ASRA	(arithmetic shift right, A)
ASLA	(arithmetic shift left, A)
ASRB	(arithmetic shift right, B)
ASLB	(arithmetic shift left, B)

LOCATION FIELD	INSTRUCTION FIELD	VARIABLE FIELD	COMMENTS
Label	Inst. Mnemonic		Variable field is blank.
Label (label is optional)	Inst. Mnemonic	Expression	The value of the expression specifies either source/destination registers and overflow conditions, a shift count, an I/O device or function, or a halt number.

DAS recognizes the following type-4 mnemonics:

TZA, TZB, TZX	(clear register)
IAR, IBR, IXR	(increment register)
DAR, DBR, DXR	(decrement register)
CPA, CPB, CPX	(complement register)
TAB, TBA, TAX, TXA, TBX, TXB	(register transfer)
SØF, RØF	(overflow)
HLT, NØP	(control)

The following instruction mnemonics are functionally the same as the preceding register change instructions except that these mnemonics allow the user to specify multiple-source and/or destination registers, or specify whether or not function execution is dependent on the overflow conditions:

INSTRUCTION MNEUMONIC	INSTRUCTION FUNCTION
MERGE	Take the inclusive-OR of the contents of all specified source registers and deliver the result to each of the specified destination registers.
CØMPL	Like MERGE, except the result is ones-complemented before delivery.
INCR	Like MERGE, except + 1 is added to result before delivery.
DECR	Like MERGE, except + 1 is subtracted from the result before delivery.
ZERØ	Zero each of the specified destination registers.

The value of the expression used in the variable field of the instruction is interpreted by DAS as having the following meaning:

If bit 0 = 1 :	A is a destination register
If bit 1 = 1 :	B is a destination register
If bit 2 = 1 :	X is a destination register
If bit 3 = 1 :	A is a source register
If bit 4 = 1 :	B is a source register
If bit 5 = 1 :	X is a source register
If bit 8 = 1 :	The function is to be performed if and only if the overflow flip-flop is set to 1.

The instruction generated by DAS has the following format. Bits 8, 5, 4, 3, 2, 1, and 0 are extracted directly from the corresponding bits of the expression value.

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	V (0)	f	f	X	B	A	X	B	A
18-bit Option									Conditional on Overflow	Function		Source Registers			Destination Registers		

The shift mnemonics recognized by DAS are listed below. The expression value represents the number of positions to be shifted. A value outside the range of 0-31 is reduced modulo 31 and an error code is printed.

LSRA	(logical shift right, A)
LRLA	(logical rotate left, A)
LSRB	(logical shift right, B)
LRLB	(logical rotate left, B)
ASRA	(arithmetic shift right, A)
ASLA	(arithmetic shift left, A)
ASRB	(arithmetic shift right, B)
ASLB	(arithmetic shift left, B)

LLSR	(long logical shift right)
LLRL	(long logical rotate left)
LASR	(long arithmetic shift right)
LASL	(long arithmetic shift left)

The following single-word input/output instructions are recognized by DAS. The expression value specifies the I/O function (EXC instruction) and device:

EXC	(external control I/O function and device)
INA	(input from the selected I/O device is inclusively-ORed with the contents of A)
INB	(input from the selected I/O device is inclusively-ORedwith the contents of B)
INAB	(input from the selected I/O device is inclusively-ORed with the contents of A and with the contents of B)
CIA	(A is cleared, then the input data is placed in A)
CIB	(B is cleared, then the input data is placed in B)
CIAB	(A and B are cleared, then the input data is placed in both registers)
ØAR	(output from A)
ØBR	(output from B)
ØAB	(the inclusive-OR of the A and B is output to the selected device)

2.4 DAS PSEUDO INSTRUCTIONS

2.4.1 General

The following set of pseudo instructions is provided to allow the DATA 620/i programmer complete control of the assembly process. The pseudo instructions are divided into the following groups:

- Label Definition
- Instruction Definition
- Location Counter Control
- Data Definition
- Memory Storage Reservation
- Conditional Assembly
- Assembler Control
- Subroutine Control
- List and Punch Controls

2.4.2 Label Definition

The label table is a list of labels that occur in the source program. To each label, there is a corresponding value, usually an address. The programmer may assign arbitrary values to labels by means of the pseudo instructions described in the following paragraphs.

2.4.2.1 EQU pseudo instruction

Location Field: a label
Instruction Field: EQU
Variable Field: An expression

The label is placed in the label table and assigned the value of the expression in the variable field. If the label is already in the label table, an error message (DD) is printed and the value of the expression replaces the value in the the table. Any label appearing in the expression must have been defined previously, for correct assembly.

2.4.2.2 SET pseudo instruction

Location Field: a label
Instruction Field: SET
Variable Field: an expression

If the label in the location field has not yet appeared in the location field in any instruction, this label is entered into the label table and assigned the value of the expression in the variable field. If the label is already in the label table the value of the expression in the variable field replaces the previous value of the label. Any labels appearing in the variable field must have been defined previously, for correct assembly.

EQU and SET are essentially the same except that redefinition of a label is permitted by SET without an error message.

2.4.2.3 MAX pseudo instruction

Location Field: a label
Instruction Field: MAX
Variable Field: two or more expressions separated by commas.

The label in the location field is assigned the greatest of the algebraic values of the expressions appearing in the variable field. All labels appearing in the variable field must have been previously defined, for correct assembly. Redefinition of the label is permitted by the SET pseudo instruction (paragraph 2.4.2.2).

2.4.2.4 MIN pseudo instruction

Location Field: a label
Instruction Field: MIN
Variable Field: two or more expressions separated by commas

The label in the location field is assigned the smallest of the algebriac values of the expressions appearing in the variable field. All labels appearing in the variable field must have been previously defined, for correct assembly. Redefinition of the label is permitted by the SET pseudo instruction (paragraph 2.4.2.2).

2.4.3 Instruction Definition

The DATA 620/i programmer may redefine a standard instruction mnemonic with the pseudo instruction OPSY.

2.4.3.1 ØPSY pseudo instruction

Location Field: a label
Instruction Field: ØPSY
Variable Field: an instruction mnemonic

The label in the location field becomes an instruction mnemonic, with the same definitions as the mnemonic in the variable field. This pseudo instruction is used to redefine the standard instruction mnemonics.

Examples:	CLA	ØPSY	LDA
		CLA	BETA

2.4.4 Location Counter Control

Five pseudo instructions are provided for controlling the DAS location counters (LC). Multiple location counters are provided in the DAS assembler, along with pseudo instructions to preset or modify the values of an individual location counter.

The following table lists the five LC labels which are standard in the DAS system. These LC labels need not be created by the DATA 620/i programmer.

IC LABEL	INITIAL VALUE	INTENDED USE
SYØE	$00001_{(8)}$	Controls the assignment of locations to any system parameters desired by the user.
IAØR	$00100_{(8)}$	Controls the assignment of locations to indirect pointers.
LTØR	$01000_{(8)}$	Controls the assignment of locations to literals.
CØMM	$02000_{(8)}$	Controls the assignment of locations within an interface area which is common to two or more programs.
Blank	$04000_{(8)}$	The blank location counter is used initially by DAS for assigning locations. This is the counter normally in use by DAS unless the programming tells it to do otherwise with USE pseudo instruction.

In addition, to the five standard location counters, the DAS programmer may create up to eight of his own location counters. This allows the programmer to create complex relocatable and overlay programs within a single assembly.

At the beginning of an assembly, there are no created location counters. DAS uses, at any time, three location counters for location assignment. The IAØR and LTØR location counters are always in use. A third location counter is used to assign locations to generated instructions and to generated data (except literals and indirect pointers). The blank location counter is initially used by DAS to control this function until another LC symbol is so designated by the pseudo instruction USE (paragraph 2.4.4.3).

For a straightforward program which uses one LC, complete control over the LC is maintained by pseudo instructions ØRG (paragraph 2.4.4.1) and LØC (paragraph 2.4.4.2).

2.4.4.1 ØRG pseudo instruction

The location counter that is currently in use is set to the value of the expression in the variable field. If a label appears in the location field, the label is set to the value in the variable field. If a label appears in the expression, the label must have been previously defined for correct assembly.

Location Field: label or blank
Instruction Field: ØRG
Variable Field: an expression

2.4.4.2 LØC pseudo instruction

The LØC pseudo instruction causes instructions and/or data following LØC to be generated as if the ØRG pseudo instruction had been used to change the current LC value. However, the value of the LC is not changed by the LØC pseudo instru-tion and the instructions and/or data generated are located in memory at the LC address.

The LØC pseudo instruction is used if the instructions and data following the LOC address are to be moved to the LØC address by the object program before execution. If a label appears in the variable field, the label should have been previously defined for correct assembly.

The LØC pseudo instruction may not be used with a relocatable program.

Location Field: label or blank
Instruction Field: LØC
Variable Field: an expression

2.4.4.3 BEGIN pseudo instruction

The BEGIN pseudo instruction allows the DAS programmer to create a new location counter or to redefine the value of any location counter before using it. The location counter is given a value equal to the expression in the variable field. BEGIN does not have any effect on the location counter currently being used.

Once a location counter has been used by a DAS program for location assignment, the value of that location counter may not be redefined by the BEGIN pseudo instruction. If a label appears in the expression in the variable field, the label must have been previously defined for correct assembly.

2.4.4.4 USE pseudo instruction

The USE pseudo instruction causes DAS to use the location counter designated in the variable field to assign locations to the instructions and data (except literal and indirect pointers) following USE.

Location Field: blank
Instruction Field: USE
Variable Field: blank, CØMM, SYØR, or a created LC label

If the variable field is the character string PREV, then the LC used previously is recalled. Only one previous usage is remembered. Thus, the sequence

```
USE A        or      USE C
USE B                USE A
USE PREV             USE B
                     USE PREV
                     USE PREV
```

are both equivalent to USE A.

2.4.5 Data Definition

2.4.5.1 DATA pseudo instruction

A data item may be a direct or indirect address constant (paragraph 2.2.5.4 and 2.2.5.5.) or it may be an expression.

If a label appears in the location field, the label is assigned to the memory location of the first generated word.

Location Field: a label or blank
Instruction Field: DATA
Variable Field: one or more data items separated by commas

2.4.5.2 PZE pseudo instruction

The PZE (plus zero) pseudo instruction is essentially the DATA pseudo instruction except that the sign bit of the data word is always set to zero (plus).

Location Field: a label or blanks
Instruction Field: PZE
Variable Field: one or more data items separated by commas.

2.4.5.3 MZE pseudo instruction

The MZE (minus zero) pseudo instruction is essentially the DATA pseudo instruction except that the sign bit of the data word is set to one (minus).

Location Field: a label or blanks
Instruction Field: MZE
Variable Field: one or more data items separated by commas

2.4.6 Memory Storage Reservation

2.4.6.1 BSS pseudo instruction

BSS causes the location counter to be increased by the value of the expression in the variable field. If a label appears in the location field, it will be assigned the value of the location counter prior to the increase in the location counter. (The location counter is always set at the address of the next available word.)

Location Field: a label or blanks
Instruction Field: BSS
Variable Field: an expression

2.4.6.2 BES pseudo instruction

BES causes the location counter to be increased by the value of the expression in the variable field. If a label appears in the location field, the label is assigned to the address value of the incremented location counter minus one.

Location Field: a label or blanks
Instruction Field: BES
Variable Field: an expression

2.4.6.3 DUP pseudo instruction

Location Field: blank
Instruction Field: DUP
Variable Field: one of three forms, as follows:

Form 1: No address fields. The instruction is ignored.

Form 2: One address field. Example: DUP, n (the next source statement is duplicated n times).

Form 3: Two address fields. Example: DUP, n, m (the next m source statements are duplicated n times where $m \le 3$, $n \le 32{,}767$.) If either field contains a zero the field will be treated as though a one were present.

2.4.7 Conditional Assembly

The following five pseudo instructions are provided to conditionally assemble various portions of a DATA 620/i program.

2.4.7.1 IFT and IFF pseudo instructions

Location Field: blank

Instruction Field: IFT or IFF

Variable Field: one, two, or three expressions separated by commas.
IFF (if false) is the logical complement of the IFT (if true) instruction

The instruction

INSTRUCTION FIELD	VARIABLE FIELD
IFT	A, B, C

means: include the next line of code if A< B and B ≤ C. The form A,,B means A ≠ B. The form A is true if A ≠ 0, otherwise false.

The following are examples of frequently used forms:

INSTRUCTION FIELD	VARIABLE FIELD	COMMENTS
IFF	A,, B	for A = B
IFT	A, B, B	for A≤B
IFT	0, A, B	for A<B and A>0
IFF	A	for A = 0

2.4.7.2 GØTØ pseudo instruction

GØTØ is used to skip more than one instruction. GØTØ, which usually follows an IFT, or IFF pseudo instruction, may not be used to jump to an earlier point in the program. All instructions following GØTØ, up to but not including the first instruction containing the designated symbol in its location field, are skipped.

The instructions that have been skipped are listed, unless suppressed by a comma following the symbol in the variable field, or the SMRY pseudo instruction (paragraph 2.4.10.7).

Label Field: blank
Instruction Field: GØTØ
Variable Field: one of forms
a) symbol
b) symbol,
c) decimal integer
d) decimal integer

2.4.7.3 CØNT and NULL pseudo instructions

The CØNT (continue) or NULL pseudo instruction provides a target for a previously appearing GØTØ. No object data is generated with the CØNT or NULL pseudo instructions. The NULL instruction will not be listed if the SMRY pseudo instruction is in effect.

Location Field: blank
Instruction Field: CØNT or NULL
Variable Field: decimal integer or label

Example:

```
        N     EQU      16
              IFT      N-16        N-16=0 (FALSE)
              GØTØ     YYY         GØ INCLUDE CØDING FØR
                                     18 BIT
*             (CØDING FOR 16 BIT)
              IFF      N-16        N-16=0 (FALSE)
              GØTØ     ZZZ         BY PASS 18 BIT CØDING
*       YYY   (CØDING FOR 18 BIT)
        ZZZ   CØNT                 CØMMON CØDING
```

2.4.8 Assembler Control

2.4.8.1 END pseudo instruction

DAS requires the END pseudo instruction as the last source statement in the program. The value of the expression in the variable field is used by the loader as the entry point into the program, after the program has been loaded into the DATA 620/i. A blank expression field designates location 00000 as the entry point.

Location Field: blank
Instruction Field: END
Variable Field: an expression

2.4.8.2 MØRE pseudo instruction

MØRE is used to inform DAS that additional inputs are to be placed in the source input device. The DAS assembly system executes a halt to allow the additional source statements to be placed in the input device. Assembly resumes when the RUN pushbutton on the computer control console is pressed. This pseudo instruction is never listed.

Location Field: blank
Instruction Field: MØRE
Variable Field: blank

2.4.9 Subroutine Control

The three pseudo instructions provided for the creation and use of closed subroutines are described in the following paragraphs.

2.4.9.1 ENTR pseudo instruction

ENTR causes DAS to assemble a closed subroutine. The label in the location field is the name of the subroutine. The ENTR generates the linage word (zero) in the object subroutine.

Location Field: label
Instruction Field: ENTR
Variable Field: blank

2.4.9.2 RETU pseudo instruction

RETU is used to return from a closed subroutine. An unconditional branch is generated to the value of the expression in the variable field.

Location Field: label or blank
Instruction Field: RETU
Variable Field: an expression

2.4.9.3 CALL pseudo instruction

If a label appears in the location field, the label is entered into the label table and assigned the present value of the (current) location counter. The first subfield must contain a valid label (the name of a subroutine). The list subfields may contain any valid DATA items (paragraph 2.4.5.1).

Location Field: a label or blank
Instruction Field: CALL
Variable Field: one or more subfields, as follows:

a. symbol (required)
b. parameter list (optional)
c. error return list (optional)

Example: , CALL, FUNC, X, Y + 1, (ERR), (GØØF)*

This produces a machine code identical to that which would be obtained by:

, JMPM, FUNC
, DATA, X, Y + 1, (ERR), (GØØF)*

2.4.10 List and Punch Controls

The following eight pseudo instructions provide the DATA 620/i programmer complete control over the listing and punching functions during program assembly. These controls are operative only during the second pass of DAS.

2.4.10.1 LIST pseudo instruction

LIST informs the DAS assembly system that a program listing is to be produced. DAS is initially in a LIST condition.

Location Field: blank
Instruction Field: LIST
Variable Field: blank

2.4.10.2 NLIS pseudo instruction

NLIS suppresses further listing of the program.

Location Field: blank
Instruction Field: NLIS
Variable Field: blank

2.4.10.3 PUNC pseudo instruction

The PUNC pseudo instruction produces an object paper tape program from the DAS assembly system. DAS is initially in a PUNC condition.

Location Field: blank
Instruction Field: PUNC
Variable Field: blank

2.4.10.4 NPUN pseudo instruction

NPUN suppresses further object paper tape output from the DAS assembly system.

Location Field: blank
Instruction Field: NPUN
Variable Field: blank

2.4.10.5 SPAC pseudo instruction

The listing device is spaced by the number of lines in the variable field. The SPAC pseudo instruction itself is not listed.

Location Field: blank
Instruction Field: SPAC
Variable Field: an expression

2.4.10.6 EJEC pseudo instruction

The EJEC pseudo instruction restores the listing device to the top of the form. EJEC itself does not appear on the listing.

Location Field: blank
Instruction Field: EJEC
Variable Field: blank

2.4.10.7 SMRY pseudo instruction

SMRY suppresses the listing of source statements which have been skipped by the condition assembly controls (paragraph 2.4.8), and the listing of the symbol table on pass 1.

Location Field: blank
Instruction Field: SMRY
Variable Field: blank

2.4.10.8 DETL pseudo instruction

DETL removes the effect of the SMRY pseudo instruction (paragraph 2.4.10.7). That is, all source statements are listed. The normal mode of operation of the DAS system is the DETL mode.

Location Field: blank
Instruction Field: DETL
Variable Field: blank

2.4.10.9 READ pseudo instruction

DAS is initially set to process up to 80 characters per line. This instruction will permit n number of characters from each source line to be processed by the assembler. If n is less than 20 or greater than 80, the number of characters read will be reset to 80 and a SZ message will be listed. A SMRY pseudo instruction will suppress the listing of READ cards during pass 2, unless there is a size error message.

Paper Tape:

Location Field: blank
Instruction Field: READ
Variable Field: n

Cards:

DAS is initialized to 026 keypunch codes

INSTRUCTION FIELD	VARIABLE FIELD	ACTION INITIATED
READ	80,29	Reads 80 columns of 029 codes, in all succeeding cards.
READ	72,26	Reads 72 columns of 026 codes, in all succeeding cards.
READ	29	Does not change number of columns read, does change type of codes.
READ	80	Reads 80 columns, does not change codes.

If the code type is not 26 or 29 the assembly will stop with A, B, X, and U registers equal to 26. At this time the card may be corrected and put back in the card reader. Pressing the RUN button will continue the assembly.

2.5 SOURCE STATEMENT FORMATS

2.5.1 Punched Card Format

When input is presented to the DAS System on punched cards, the following format rules apply. A symbolic card consists of four fields: location field, instruction field, variable field, and remarks field.

2.5.1.1 Location Field: This field is used to attach a label name or a target number (refer to the GØTØ pseudo instruction, paragraph 2.4.7.2) to a source statement. Use of the location field is optional, but if used, the label or number must begin in column 1 and must not extend beyond column 6 of the punched card.

2.5.1.2 Instruction Field. The instruction field, beginning in column 8, holds a mnemonic representing the computer instruction or a DAS pseudo instruction. This field must not extend beyond column 14. Indirect addressing is indicated by an asterisk(*), following the instruction mnemonic.

2.5.1.3 Variable Field. The variable field begins in column 16 and ends with the first blank which is not contained within a character constant. The contents of the variable field vary according to the instruction and will normally consist of one or more subfields, separated by commas. The variable field is not required for all instructions.

2.5.1.4 Remarks Field. The remainder of the card, following the variable field, if present, or starting in column 17, may be used for commentary. This field is ignored by the DAS, but will appear on the listing.

2.5.1.5 Comments statement. An entire source card may be used for commentary by placing an asterisk as the first non-blank character in the location field. The contents of the statement will be ignored by DAS, but will appear on the output listing.

2.5.2 Paper Tape Format

An alternative, column-independent, imput form is provided by punched paper tape, which may be conveniently prepared on the Teletype. The term "code line" will be used within this section instead of "symbolic card", to indicate a source statement on paper tape.

2.5.2.1 Paper tape code line. The maximum length of the code line, in the DAS system, is 80 characters, plug the line feed characters.

Location Field		Instruction Field		Variable Field		Remarks Field	Carriage Return/ Line Feed
					b		

The carriage return (CR) character should be used preceding the line feed (LF) character for typeout control.

2.5.2.2 Location Field. The location field may contain a label, an extended symbol, or a target number. The first four non-blank characters are used as the label. The location field is void if the first non-blank character of the code line is a comma.

2.5.2.3 Instruction Field. The instruction field may contain a mnemonic, or a mnemonic followed by an asterisk (*) which indicates indirect addressing.

2.5.2.4 Variable Field. The variable field may contain one or more subfields separated by commas. The variable field is terminated by either a blank (which is not part of a character constant), a CR or a LF. Each subfield may contain an expression or a constant of any type, or may be voided by using adjacent commas.

2.5.2.5 Remarks Field. The remarks field consists of any text between the terminating blank of the variable field and the next CR or LF character and is ignored by DAS.

2.5.2.6 Comments Line. If the first non-blank character on a code line is an asterisk (*), the entire line is ignored by the DAS system, but will appear on the output listing.

2.6 DAS OUTPUT LIST

2.6.1 DAS Source Listing

The DAS assembly system allows the programmer to obtain an on-line listing of his program, either in parts, or the entire program, as the program is being assembled. The symbolic (source) program and the object (absolute) program are listed side-by-side on the listing device (either teletype or printer).

Error analysis is performed during assembly and, as errors are detected, error codes (paragraph 2.6.2), are printed on the line following the source/object information.

The list controls pseudo instructions: LIST, NLIS, SPAC, EJEC, SMRY, and DETL are described in paragraph 2.4.10 and subparagraphs.

The format of the data on the output listing is:

LOCATION	OBJECT CODE	ADDRESS MODE	SOURCE STATEMENT	COMMENTS
014000			, ØRG , 014000	
014000	000000		ABS , ENTR ,	
014001	001002		, JAP* , ABS	
014002	114000	R		
014003	005211		, CPA ,	
014004	001000		, JMP* , ABS	
014005	114000	R		
	000000		, END ,	

Address modes include:

C - FORTRAN common reference.
E - externally defined.
I - indirect pointer.
R - absolute/relative.

2.6.2 DAS Error Messages

The DAS assembly system performs extensive syntax checking during both passes of the assembler. During the first pass, detectable errors are listed. When an error is detected on the second pass of DAS, the following information is listed:

- Error code

- Value of location counter

- Object code when instruction has been assembled unless a NLIS pseudo instruction (paragraph 2.4.10.2) is in effect or a list suppress comma is present on a GØTØ pseudo instruction (paragraph 2.4.7.2).

Up to four error messages may occur on a line of output listing. The error message is preceded by a list of the source statement.

The following error codes are produced by DAS:

CODE	MEANING
*IL	The first non-blank character on a line is illegal, line not processed.
*ØP	The instruction code is undefined; a two-word gap is left in memory to allow patching.
*SY	Expression contains an undefined label.
*EX	Expression contains the illegal appearance of two consecutive arithmetic operators.
*SP	Illegal use of a special character for operand in address evaluation.
*AD	Address expression in error.
*FF	Floating-point format error.
*DC	A decimal character appears in an octal constant.
*DD	Illegal redefinition of a lable or location counter.
*VF	Instruction contains variable subfields either missing or inconsistent with the computer instruction type.
*MA	Inconsistent use of indexing and indirect addressing.
*XR	Address out of range for index specification.
*NS	Nested DUP statements.
*NR	No room left in label table for this label.
*TF	Tag error, undefined or illegal index.
*=	Illegal use of literal =.
*SZ	Expression value too large for size of subfield.
*UD	Undefined label in variable field of a USE instruction.

CODE	MEANING
*CH	Illegal character in source line.
*QQ	Illegal use of quotation marks.

2.7 OPERATING THE DAS ASSEMBLY SYSTEM

The assembler tape is loaded into memory using the binary load program (see section III). After the assembler loading is complete the normal system input, output, and listing devices are readied, the sense switch(es) are set depending on pass. Sense switch 1 for pass 1 and sense switch 2 and 3 for pass 2. To begin assembly, RUN at location 000001.

Termination of pass 1 and 2 is initiated whenever an END pseudo-op is detected. END causes a HALT 0777 to be executed with the A, B and X registers set to -1 (all ones). To initiate pass 2, reset the I/O devices, set the sense switches, and RUN. Pass 2 may be repeated as often as desired to produce extra copies of the program.

The computer will execute a HALT 0777 when a MØRE pseudo-op is detected, and display 0170017 (octal) in the A, B and X registers. Prepare the input units and RUN. Synchronization errors are detected on pass 2 when the address value of a label does not agree with the value assigned on pass 1. Synchronization errors are due to misreads of the source tape and cause DAS to halt with the A, B, and X registers set to 0777. To continue the assembly process, press RUN. The assembler will reset the location counter to the value assigned during the 1st pass, print the error message SE, and continue.

2.8 FORTRAN PSEUDO INSTRUCTIONS

2.8.1 General

The following special op codes are provided for the DATA 620/i programmer in order to provide assembly output compatible with the FORTRAN loader.

2.8.1.1 FØRT op code. This op code must be the first line of code in an assembly, except for comments. It indicates that the output must be compatible with the FORTRAN relocatable loader.

2.8.1.2 NAME op code. This op code must be the second line of code in an assembly, except for comments. It contains the name of the entry point in the address field. The label field is left blank. The name indicated is provided to the assembly program and output for the loader in order to allow linkage to the routine from other routines. Multiple entry points are allowed.

2.8.1.3 CØMN op code. This op code is used to define common areas. The area name is placed in the label field and the length in the address field. This op code may be placed anywhere within the program. Only one name is defined for each use of the op code, and the names and area lengths are cumulative. It has approximately the same effect as a series of BSS instructions, except the area is defined to be in the common pool.

2.8.1.4 EXT op code. This op code is used to indicate that a symbol is not undefined, but resident in another routine. The symbol to be so identified is placed in the label field. The address field is unused. One such symbol can be defined with each use of this op code. This code may be placed at any point within the program.

2.8.2 Relocation

In order to allow relocation, the system requires that all one word instructions that address locations in memory use the relative forward method of addressing. All two word instructions are legal.

2.8.3 Literals

No literals may be used. The use of immediate instructions is recommended.

2.8.4 Restrictions

All expressions containing symbols defined with CØMN, or EXT instructions must be the second word of two word instructions, or part of a DATA, PZE or MZE instruction.

The FORTRAN compiler uses two words for each value retained in core. For this reason it is necessary for the assembly language writer to make allowance for this when defining CØMMØN.

If FORTRAN had the statement: CØMMØN A(4), B(3, 4) DAS should have:

```
A     ,    CØMN , 4*2
B     ,    CØMN , 3*4*2
```

2.8.5 Modes

All symbols and expressions are given a mode. This mode is either external, common, relative, or absolute. The definition of the mode is assigned by the assembler according to certain rules. Both symbols and expressions have a mode. The mode of an expression is determined by the mode of the symbols used within the expression.

The mode of a symbol is defined as follows:

If the symbol is defined with the EXT op code the mode is E.

If the symbol is defined with the CØMN op code the mode is C.

If the symbol is a numeric constant the mode is A.

If the symbol is * used as the current location the mode of the * is R.

If the symbol is defined by an EQU, SYN etc. the mode is that of the expression on the right side of the op code.

If the symbol is a label in a program the mode is R.

The mode of an expression is assigned as follows:

If the expression contains any symbol of mode E the expression is mode E.

If the expression contains any symbol of mode C the expression is mode E.

If the expression contains only mode A symbols the expression is mode A.

If the expression contains A and R symbols the mode is R if an odd number of mode R symbols appear, otherwise the mode is A.

Certain restrictions appear within the DAS assembler when providing FORTRAN compatible output. The restrictions on expressions are:

No expression may contain both mode E and C symbols.

Any type E expression must consist only of the type E symbol.

No type E, C or R expression should include the multiplication or division of a type E or C symbol.

No expression should contain the sum or difference of a mode C symbol and a mode R symbol, or a mode E symbol and a mode R symbol.

No expression should contain the sum of two mode E, C or R symbols.

A mode A symbol may be added to or subtracted from a mode C or R symbol.

Examples:

```
EEEE  , EXT   ,                      EEEE defined as type E
CCCC  , COMN  , 6                    CCCC defined as type C
RTN   , ENTR  ,                      RTN is type R, a label
TBL   , BSS   , 50                   TBL is type R
ABL   , BSS   , 'A'+5                ABL is type R
LENG  , EQU   , *-TBL                LENG is type A, length of area
      , CALL  , EEEE, TBL, LENG
      , LDA   , *+6                  Ok, relative forward
      , LDA   , CCCC+6               Illegal, one word inst, not R or A
      , LDXI  , CCCC+6               Ok, two word instruction
      , LDA   , 0, 1                 Get CCCC+6 to A, legal
      , DATA  , EEEE+4               Illegal, value not zero
      , DATA  , CCCC+4               Legal
      , DATA  , CCCC+LENG            Legal
      , DATA  , TBL+LENG-5           Legal, mode is R
```

2.8.6 Example of FORTRAN Compatible Assembly

```
        000000  R         , FORT   ,
        000017  R         , NAME   , $PE
        000000  E   $SE   , EXT    ,
        000000  E   $QS   , EXT    ,
        000000  E   $QE   , EXT    ,
000000  074025            , STX    , $PE+7
000001  034021            , LDX    , $PE+4
000002  054025            , STA    , $PE+9
000003  064025            , STB    , $PE+10
000004  015000            , LDA    , 0, 1       PAR.I
000005  034020            , LDX    , $PE+7
000006  002000            , JMPM   , $QS
000007  000000  E
000010  000026  R         , DATA   , $PE+7
000011  014016            , LDA    , $PE+9
000012  024016            , LDB    , $PE+10
000013  002000            , JMPM   , $QE        A**B SUBROUTINE
000014  000000  E
000015  000026  R         , DATA   , $PE+7
000016  001000            , JMP    , 0
000017  000000
000017                    , ORG    , *-1
000017  000000      $PE   , ENTR   ,
000020  002000            , CALL   , $SE, 1
000021  000000  E
```

```
000022  000001
000023  000000            , DATA      , 0
000024  001000            , JMP       , *-20
000025  000000   R
000026  000000            , DATA      , 0,0,0,0
000027  000000
000030  000000
000031  000000
        000000            , END       ,
```


SECTION III

AID-UTILITY AND DEBUGGING PACKAGE

3.1 INTRODUCTION

These programs are a collection of useful diagnostic and utility routines for the DATA 620/i computer. The operator can call upon a wide variety of functions to aid him in debugging and running his programs. Specifically these programs are:

(1) Bootstrap loader program
(2) Binary load dump
(3) AID

3.2 BOOTSTRAP LOADER

This program is typically used when a "cold start" is required. A cold start usually occurs when the specific contents of memory is not known to the operator.

The procedure for loading the program is shown below. Use only those procedures which apply to specific system configuration.

(1a) Turn on paper tape reader.

(1b) Turn on model 33A teletype.

(1c) Place model 33/35B teletype in off-line mode and press control and D, T, and Q to initialize teletype.

1. Position the tape in the reader with the first binary frame at the read station.

2. Set the reader control lever in the STOP or LOAD position and set the teletype on-line. For paper tape reader, no action required.

3. Enter the appropriate bootstrap load routine into memory through the console. See below.

4. Set A = B = 0, IC = X7770, X = X7600, press SYSTEM RESET and RUN.

5. To initiate loading, set the reader control lever in the START or RUN position.

6. A successful load of the loader and punch program is indicated by a halt at X7600 with B = 0 and the reader halted.

7. Common causes for failure are:

a. The proper bootstrap load routine was not in memory.
b. The bootstrap was not positioned correctly.
c. The registers were not set correctly.
d. The teletype was not 'on-line'.

DATA 620/i BOOTSTRAP LOAD ROUTINES

LOCATION	HIGH SPEED READER	MODEL B TELETYPE	MODEL A TELETYPE	SYMBOLIC
X7756	102637	102601	102600	READ, CIB, RDR
X7757	004011*	004011*	004011*	, ASLB, NBIT-7
X7760	004041	004041	004041	, LRLB, 1
X7761	004446	004446	004446	, LLRL, 6
X7762	001020	001020	001020	, JBZ, SEL
X7763	0X7772	0X7772	0X7772	
X7764	055000	055000	055000	, STA, 0, 1
X7765	001010	001010	001010	, JAZ, LHLT + 1
X7766	0X7600	0X7600	0X7600	
X7767	005144	005144	005144	, IXR,
X7770	005101	005101	005101	ENTR, INCR, 1
X7771	100537	102601	100000	SEL , SEL, RDØN
X7772	101537	101201	101100	, SEN, IBFR, READ
X7773	0X7756	0X7756	0X7756	
X7774	001000	001000	001000	, JMP, *-2
X7775	0X7772	0X7772	0X7772	

*For 18-bit computers insert 4013.
X = 0 for a 4K memory, X = 1 for an 8K memory, etc.

This example would result in the first element of common being the integer variable I; the next five elements of common being the real vector array A; and the next element in common being the real variable B.

3.4 EQUIVALENCE STATEMENT

Form: EQUIVALENCE (k), (k_2), . . ., (k_n), where each (k) is a list of two or more non-dummy variables and/or array element names, separated by commas. Subscript expressions of array element names must be non-zero, unsigned integer constants. A two dimensional array may be referred to by using a single subscript, giving the element number within the array, if desired.

The effect of the EQUIVALENCE statement is to cause the came area of memory to be shared by two or more entities. Each element of the K_i list is assigned the same (or a part of the same) storage area.

More than one EQUIVALENCE statement is permitted in a program, but it may only be preceded by a SUBROUTINE, FUNCTION, DIMENSION, COMMON or prior EQUIVALENCE statement.

Example:

```
DIMENSION      A(5), I1 (3,3), B1(3)
COMMON         B, B1, B2
EQUIVALENCE    (X,A (2),Y), (B, C2, F5), (I1 (5), B2)
```

The effect of an EQUIVALENCE statement upon common assignments, may be the lengthening of common. This lengthening is permitted only if it increases common in the same direction as additional common elements would. Thus, in the example, the equivalence (B, I1 (5)) would have been invalid. It is also invalid to equate two elements of the same array to each other.

3.3.2 Procedure to Punch Program Tapes

1. Initialize the paper tape punch and/or set the teletype 'on-line'.

2. Set the A register to the address of the first word to be punched. Set the B register to the address of the last word to be punched. Set the X register to the address of the first instruction to be executed (at load time).

3. Set the instruction counter = X7404 press SYSTEM RESET and RUN.

4. The specified memory locations will be punched and the computer will halt at X7404 with the original parameters in the registers.

5. To punch noncontiguous memory areas, set the X register to -1 (177777) for all but the last area to be punched.

3.3.3 Procedure to Punch the Bootstrap Loader

1. Initialize the paper tape punch and/or set the teletype 'on-line'.

2. Set the instruction counter = X7400, press SYSTEM RESET and RUN.

3. The loader bootstrap will be punched and the computer will halt at X7404.

4. The binary punch routine is punched following the bootstrap by setting A = X7400, B = X7600, X = 00000, and press RUN.

3.4 AID II PACKAGE FOR THE DATA 620/i

1. To enter set IC = 0X6000, where X = 0 for 4 K memory, X = 1 for 8K memory, etc., and press RUN.

2. Three pseudo registers A, B, and X are used. These registers are loaded by teletype control, or trap return. The corresponding machine registers are loaded with the pseudo register values before any GØTØ or trap command is executed.

3. Commands consist of a command letter (mnemonic) followed by a string of octal parameters, separated by commas and terminated by a period. In the description that follows, @ indicates a carriage return/line feed type-out, upper case letters are command mnemonics (A), lower case letters are octal parameters (a), letters enclosed in parentheses denote the contents of the designated location. Underlined symbols denote AID II type-outs, all others are operator entries. A parameter preceded by a minus - indicates a negative parameter.

4. AID II Commands:

A (A) . @	(Display value of pseudo A.)
B (B) . @	(Display value of pseudo B.)
X (X) . @	(Display value of pseudo X.)
A (A) a . @	(Change value of pseudo A to a.)
B (B) a . @	(Change value of pseudo B to a.)
X (X) a . @	(Change value of pseudo X to a.)
G a .	(Preset A to (A), B to (B), X to (X) and go to location a.)
T a , b . @ a (a) (A) (B) (X) @	(Preset registers and go to location b. If and when location a is reached, save and type location a, the contents of a, and the current values of the registers. (Trap to a from b.))
T a , . @ a (a) (A) (B) (X) @	(Continue trap from last breakpoint location to new breakpoint location a. (Present and save registers as before.))
I a , b , c , . @	(Initialize locations a through b (set to c).)
S a , b , c , . @ L (L) @	(Search locations a through b for words equal to c. Type out the location (L) and contents of each word thus found.)
S a , b , c , d . @ L (L) @	(Search locations a through b for words equal to c. Parameter d is used as a mask (comparison is made only for those bit positions in memory which have ones in the corresponding bits of the mask).)
S a , b , 0 , 0 . @	(Print the contents locations of a through b. (Search a through b for zero with a zero mask, no bits selected.))

a (a) @

a + 1 (a + 1) @

a + 2 (a + 2) @

•
•
•

b - 1 (b - 1) @

b (b) @

C a . @	(Change/display memory from location a.)
a (a) , @	(Display next location (a + 1).)
a + 1 (a + 1) b , @	(Change a + 1 to value b and display next location.)
a + 2 (a + 2) . @	(Quit (return to AID II).)

WARNING

An incomplete trap (no return to AID II) will leave the object program with the instruction at the breakpoint location changed to a return jump to the AID II trap return. These locations should be restored to their original values before further use of the trap function to ensure proper results.

SECTION IV

SOURCE TAPE CORRECTION PROGRAM

4.1 INTRODUCTION

The DATA 620/i symbolic correction program (COR) provides the DATA 620/i programmer a convenient method of adding or deleting source statements on symbolic paper tapes, greatly reducing program preparation time. COR eliminates the task of either completely repunching or correcting the paper tape off-line.

A statement (source statement) representing a complete line of information necessary to compile or assemble an instruction is called a "code line". The maximum length of the code line in the DATA 620/i programming system is 52 characters, plus the line feed character. The code line, the basic quantity in the COR system, may contain any character except the line feed character, and be reproduced, deleted, or replaced. In addition, a new code line (or lines) may be inserted into the program for complete up-dating capability.

4.2 OPERATING PROCEDURES FOR COR

4.2.1 Loading the COR Correction System

Loading procedures are the same for all object paper tapes punched in AID format (three bits per frame). Load the paper tape in accordance with procedures outlined in paragraph 2.7, section II of this manual.

4.2.2 Running the COR Correction System

After the COR program has been loaded into the DATA 620/i memory, place the source paper to be corrected in the ASR-33 or ASR-35 teletype paper-tape reader. Set the DATA 620/i instruction counter display to the COR symbolic location SENT+1. Before pressing the RUN pushbutton, set sense switches 1 and 2 to the desired condition.

Sense switch settings have the following meaning:

Sense Switch 1: Off - the next code line read by COR will be reproduced. On - the next code line (source statement) read by COR will be deleted and not reproduced on the updated source tape.

Sense Switch 2: Off - the computer halts between code lines, allowing the DATA 620/i programmer to insert new code lines into his program. After all of the new code lines have been added, the RUN pushbutton on the DATA 620/i is pressed and the next code line for the paper tape being updated is read into the computer.

On - the computer does not halt between code lines.

It can be seen that it is possible to delete, insert, and replace code lines by using combinations of settings of sense switches 1 and 2. Each statement punched onto the upda updated paper tape is listed on the teletype, providing the programmer with a listing of his updated program.

Observe the following rules during operation of the COR system:

- Each code line that is inserted into the updated program should begin with the carriage return and line feed characters.

- The setting of sense switch 1 should only be changed when the DATA 620/i is in the halt condition.

- The setting of sense switch 2 may be changed at any time during operation of the COR system.

- Sense switches 1 and 2 should not both be on at the same time.

A halt will occur at the end of the source paper tape being updated, regardless of the setting of sense switch 2.

FORTRAN REFERENCE

SECTION I

BASIC FORTRAN CONCEPTS

1.1 INTRODUCTION

FORTRAN is a universal, problem oriented programming language designed to simplify the computer solution of mathematical and engineering problems. The syntactical rules for the use of the language are rigorous and require the programmer to reduce the solution characteristics of his problem to a series of precise statements. These statements are evaluated and interpreted by a system program (called the FORTRAN processor) and are translated into the execution language of the computer system.

The variations between computer systems is responsible for the development of many versions of the FORTRAN language. This condition affects the number, form and relationship of the statements acceptable to a given FORTRAN processor. It is essential, therefore, that the programmer be familiar with the language specifications for the system of intended use. DATA 620/i series FORTRAN conforms with the proposed American standards for basic FORTRAN, as published by the American Standards Association on 10 March 1965.

This manual is intended for use in DATA 620/i series FORTRAN programming training classes or seminars, and as a reference for experienced programmers using the DATA 620/i series FORTRAN system.

1.2 CHARACTER SET

A FORTRAN program unit is written using the following letters, digits, and special characters:

Letters: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Digits: 0 1 2 3 4 5 6 7 8 9

Special Characters:

	(blank or space)
=	(equals)
+	(plus)
-	(minus)
*	(asterisk)
/	(slash)
(	(left parenthesis)
)	(right parenthesis)
,	(comma)
.	(decimal point)

With the exception of the specific uses indicated in the following sections of this manual, a blank character has no meaning, and may be used freely by the programmer to improve the readability of the FORTRAN program.

The following special characters are classified as arithmetic operators and are significant in the unambiguous statement of arithmetic expressions:

+	(addition or positive value)
-	(subtraction or negative value)
*	(multiplication)
/	(division)
**	(exponentiation)

The special characters equals (=), open parenthesis ((), closed parenthesis ()), comma (,) and decimal point (.), have specific application in the syntactical expression of the FORTRAN statement. The following sections of this manual will qualify their use in particular statements and expressions.

In addition to the FORTRAN character set, the DATA 620/i series FORTRAN system will accept the following characters in Hollerith fields:

S, !, ", #, %, &, ', :, ;

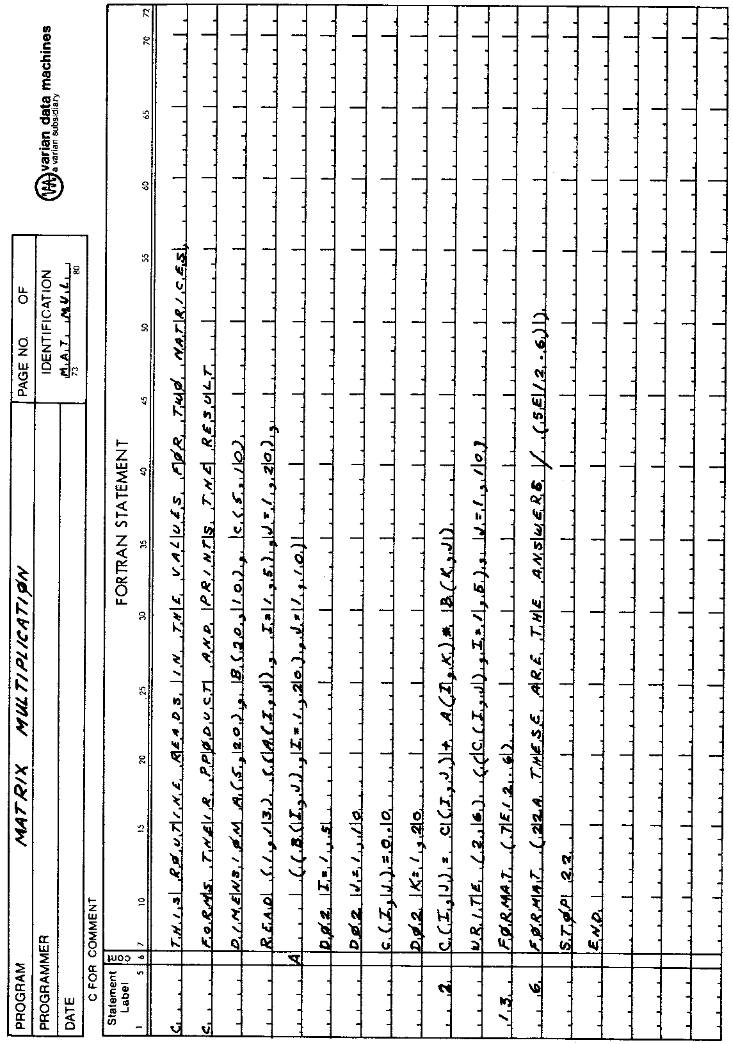
1.3 LINE FORMAT

A FORTRAN program consists of a series of statements divided into physical sections called lines, that must be coded to a precise gramatical format. FORTRAN statements fall into two broad classes, executable and non-executable. Executable statements specify program action, while non-executable statements describe the use of the program, the characteristics of the operands, editing information, statement functions, or data arrangement. The statements of a FORTRAN source program are normally written on a standard FORTRAN coding form.

Figure 1-1 is a sample FORTRAN coding form. The coding form includes 80 columns of information. Columns 73 through 80 are reserved for sequencing information, and have no effect upon the generated execution program. Columns 1 through 72 contain line information in the following format:

1.3.1 Initial Line

The first line of each statement is called an initial line. A statement may include an initial line and continuation lines. Statements may have as many continuation lines as required subject to the following restrictions: DO statements must be wholly contained on an initial line; and the equals character (=) of a replacement statement must appear on the initial line. An initial line may contain a statement label in columns 1



varian data machines
a varian subsidiary

PROGRAM MATRIX MULTIPLICATION
PROGRAMMER
DATE
PAGE NO. OF
IDENTIFICATION MAT MUL

```
C     THIS ROUTINE READS IN THE VALUES FOR TWO MATRICES
C     FORMS THEIR PRODUCT AND PRINTS THE RESULT
      DIMENSION A(5,20), B(20,10), C(5,10)
      READ (1,13) ((A(I,J), I=1,5), J=1,20),
     A ((B(I,J), I=1,20), J=1,10)
      DO 2 I=1,5
      DO 2 J=1,10
      C(I,J)=0.0
      DO 2 K=1,20
    2 C(I,J)= C(I,J)+ A(I,K)* B(K,J)
      WRITE (2,6) ((C(I,J), I=1,5), J=1,10)
   13 FORMAT (7E12.6)
    6 FORMAT (22H THESE ARE THE ANSWERS / (5E12.6))
      STOP 22
      END
```

Figure 1-1. Sample FORTRAN Coding Form.

through 5. In this case, column 6 must contain a zero digit, blank or space character; and columns 7 through 72 may contain all or part of a statement with the exception of the restrictions noted.

EXAMPLE:

```
      A=.5*C*D
```

1.3.2 Continuation Line

Continuation lines are used when additional lines of coding are required to complete a statement originating on an initial line. There may be any number of continuation lines per statement with the exceptions previously noted for initial lines. In a continuation line, columns 1 through 5 are ignored and should, but need not be blank; column 6 must contain any character other than a zero digit, blank or space character; and the continued segment of the statement is contained in columns 7 through 72. Continuation lines may only follow an initial line or another continuation line.

EXAMPLE:

```
      A=
     1.5*
     2C+
     3D
```

1.3.3 Comments Line

Any line with the character C in column 1 is identified as a comment line. Comments may appear anywhere in a program, except immediately before a continuation line. All comments lines are ignored by a FORTRAN processor, except for display purposes. Comments may be contained in columns 2 through 72.

EXAMPLE:

```
C     THIS IS A COMMENTS LINE
```


1.3.4 End Line

Any line not containing the letter C in column 1 and having only the character string END in columns 7 through 72 is recognized by the processor as an end line. Each FORTRAN program requires an end line to inform the processor that it has reached the physical end of that program.

EXAMPLE:

```
      END
```

1.3.5 Statement Label

Labels permit statements to be referenced by other portions of a program. A statement label is an integer value in the range 1 to 9999 (leading zeros or blanks are not significant for label identification). The initial line of each statement may be given a unique label in columns 1 through 5. The same label may not be given to more than one statement in a program unit.

EXAMPLE:

```
   50 A=.5*C+D
60    A=.5*C+D
 879  A=.5*C+D
```

SECTION II
DATA

2.1 GENERAL

Numerical quantities, constants and variables are distinguished in FORTRAN as a means of identifying the nature and characteristics of the numerical values encountered in program execution. A constant is a quantity whose value is explicitly stated. A variable is a numerical quantity referenced by name, rather than by its explicit appearance in a program statement. During the execution of a program, a variable quantity may assume many different values.

2.2 DATA TYPES

The DATA 620/i series FORTRAN processor recognizes two types of data, integer and real. Integer data are precise representations of integral values within the range -32767 to +32767 ($-2^{15} + 1$ to $2^{15} - 1$). Real data are approximations of real numbers with magnitudes in the range 0.588×10^{-38} to 0.588×10^{38} (approximately) 2^{-127} to $2^{127} \times (1-2^{-22})$). Both integer and real data may assume positive, negative, or zero values. The value zero is considered neither positive nor negative.

2.3 DATA NAMES

FORTRAN Data (constants, variables, arrays and array elements) are identified by names.

2.3.1 Symbolic Names

Symbolic names are made up of letter or digit strings consisting of 1 to 5 characters. The first character of the string must be a letter. Data identified by symbolic names are specified as being of type integer or real by the unique classification associated with the first letter of the character string. Names beginning with the letters I, J, K, L, M, and N are type integer; and the names beginning with any other letters are type real.

Examples of type integer symbolic names are:

I 12A MZXF N5

Examples of type real symbolic names are:

A B2 F5M79 AAA

2.4 VARIABLES

Variables are data whose values are derived and defined during program execution, and are identified by symbolic names of the appropriate type, real or integer.

2.5 CONSTANTS

Constant data are identified explicitly by naming their actual values. Constants do not change in value during program execution, and are specified to be of type integer or real.

2.5.1 Integer Constants

An integer constant is identified by a non-empty string of from 1 to 5 decimal digits written without a decimal point and optionally preceded by a plus (+) or minus (-) sign character.

Examples:

-217	-32767	+00327	512

2.5.2 Real Constants

A real constant may consist of 1 to 7 significant digits and may be identified in any one of the following forms:

±i.	±.f	±i.f	
±i.E±e	±.fE±e	±i.fE±e	±iE±e

where i, f and e are each a string of decimal digits representing an integer, fraction and exponent respectively. The plug (+) and minus (-) sign characters are optional, and the decimal point (.) and E characters are present in that form. If r represents any of the forms preceding E±e, i.e., rE±e, then the real constant is interpreted as r*10±e.

Examples:

17.	-25.620E-1	0.0	-51E1
+.42	-.479	-479E-3	.35E02

If a real constant is specified with more significant digits than the precision real data allows, truncation occurs, and only the most significant digits within the range will be represented.

2.6 ARRAYS

An array is an ordered set of data in 1 or 2 dimensions identified by a symbolic name. An array declarator (see DIMENSION Statement) defines the name and size of the array. An array name serves to identify all of the elements in the array, including data type, real or integer. An array name cannot be used without a subscript, except in Input/Output lists.

2.6.1 Array Element

An array element is one member of an array and is identified by a subscript appended to the array name.

2.6.2 Subscripts

A subscript follows the array name and contains 1 or 2 subscript expressions enclosed in parentheses. The number of subscript expressions (except in EQUIVALENCE Statements) corresponds to the specified dimensionality of the array. Two expressions within the parentheses must be separated by a comma. Subscript expressions are type integer in one of the following forms:

```
c*v±k
c*v
v±k
v
k
```

where c and k are integer constants and v is an integer variable.

Examples:

X(2*J-3)	A(I,J)	B(20)	C(L-2)

2.6.3 Dimensionality

Arrays are stored column-wise in ascending memory locations. Therefore, a 2 dimension array, A, with three rows and three columns would be stored internally in the computer as follows:

Location	Element
L+0 & L-1 *	A(1, 1)
L+2 & L+3	A(2, 1)
L+4 & L+5	A(3, 1)
L+6 & L+7	A(1, 2)
L+8 & L+9	A(2, 2)
.	.
.	.
.	.
L+16 & L+17	A(3, 3)

The position of an array element, A(i, j) is derived from the following formula:

$$A_o + (i-1 + I \times (j-1)) \times 2$$

where A_o is the location of the first element in the array; i and j are the specified row and column subscript expressions; and I is the number of row elements defined in the array declarator for A. In the example preceding, the position of the A(2,2) element would be solved in the following form:

$$L + O + (2-1 + 3 * (2-1)) \times 2 = L + 8$$

The processor collects all constant terms in subscript expressions into the base address of the referenced array.

*In DATA 620/i FORTRAN, a storage unit for a real or integer entity is two words in length.

SECTION III
SPECIFICATIONS AND STATEMENTS

3.1 GENERAL

Specification statements organize and classify data that will be referred to by other statements in the FORTRAN program. Specification statements include:

DIMENSION: Names and declares the size of an array.

COMMON: Assigns variable and/or named arrays to common storage areas.

EQUIVALENCE: Assigns variables and names arrays to shared storage areas.

Specification Statements must appear in the FORTRAN program in the order of: DIMENSION Statements, COMMON Statements and EQUIVALENCE Statements.

Examples:

Valid	Invalid
DIMENSION D(3) COMMON A, B, C, EQUIVALENCE (B, D(3))	COMMON A, B, C DIMENSION D(3) EQUIVALENCE (B, D(3))

3.2 DIMENSION STATEMENT

Form: DIMENSION $v_1(i_1)$, $v_2(i_2)$, . . ., $v_n(i_n)$, where each v(i), (called an array declarator), is composed of a declarator name v, (the name of the array), and a declarator subscript (i). Each (i) is an unsigned integer constant or two unsigned integer constants separated by a comma. Each constant must have a value greater than zero and less than the limit of available memory.

A DIMENSION statement specified that the declarator names listed are arrays in the program unit. The number of dimensions and the maximum size of each dimension is specified by the declarator subscript associated with each declarator name.

More than one DIMENSION statement may appear in a program, but can only be preceded by a FUNCTION, SUBROUTINE, or a previous DIMENSION statement.

An array element is referred to by the array name qualified by a subscript to identify the desired element. If the value of this subscript is out of the range specified by the array declarator, the derived computational results will be unpredictable.

Array elements are stored column-wise in computer memory from low address storage to high address storage. Therefore, one dimension arrays are stored sequentially in the order A_1, A_2, . . ., A_n, while two dimension arrays are stored with the first (leftmost) dimension varying most rapidly, i.e., $A_{1,1}$; $A_{2,1}$, . . ., $A_{m,1}$, $A_{1,2}$, $A_{2,2}$, . . ., $A_{m,n}$.

Example:

DIMENSION A(5), I1(3,6), C(5,10)

This specification statement indicates that A is a real vector with 5 elements; I1 is an integer matrix of size 3x6=18 elements; and C is a real matrix of size 5x10-50 elements.

3.3 COMMON STATEMENT

Form: COMMON a_1, a_2, . . ., a_n, where each a is a non-dummy variable or array name.

A COMMON statement specifies that the variables and/or arrays listed are to be assigned to storage in the memory region called COMMON. The elements named are assigned storage relative to the common origin in the order of their appearance in the COMMON statement of each program unit. By making use of this positional relationship, more than one program unit in an executable program may reference the same data directly.

Each entity type (real or integer) is assigned two storage locations relative to the beginning of common, and entities of the same type in corresponding position are the same quantity. Entities referenced by position are the correct type, if the most recent value assignment to that position was of the same type.

The size of common in each program unit of an executable program may vary without disturbing the specified positional relationship. The beginning of common is established during the loading process with the program unit with the largest common region and all other program units are adjusted to begin at this location.

A program may have more than one COMMON statement, however, it may be preceded only by a FUNCTION, SUBROUTINE, DIMENSION or a prior COMMON statement.

Example:

```
DIMENSION    A(5)
COMMON       I, A, B
```


3.3 BINARY LOAD/DUMP

These programs are distributed in object form on a single tape labeled binary load dump. The binary load program is in a special format called bootstrap format and the dump program is in standard binary format.

Essentially what happens is as follows:

1. Using bootstrap loader (discussed in previous section) the binary loader is loaded into memory.
2. Upon completion of the load process, control is transferred to the binary loaded (recently unloaded) and;
3. It then loads the binary dump program into memory.

3.3.1 Procedure to Load Program Tapes

1. Initialize the paper tape reader and/or set the teletype 'on-line'.
2. Place the program tape in the reader and place the reader control lever in the RUN position.
3. Set the A register to the load mode:
   - <0 to verify the program tape
   - = 0 to load the program tape and halt
   - > 0 to load the program tape and execute the program
4. Set the instruction counter = X7600, press SYSTEM RESET and RUN.
5. A successful load is indicated by a halt at X7600 with the A register set to the load mode, the B register set to 0, and the X register set to the execution address.
6. A checksum or format error causes a halt at X7600 with the B register set to -1 (177777) and the X register set to the load address of the last record read.
7. To restart, position the program tape at the previous record mark and press RUN.

SECTION IV

EXPRESSIONS AND ASSIGNMENTS

4.1 ARITHMETIC EXPRESSIONS

An arithmetic expression is formed in FORTRAN syntax by a combination of operations and elements. The expression and its elements identify the expression to be type integer or real.

The arithmetic operators are shown in the following table:

OPERATOR	FUNCTION
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The arithmetic elements are described by the following statements:

Primary. An arithmetic expression enclosed in parenthesis, a constant, a variable reference, an array element reference or function reference.

Factor. A factor is a primary of the forms:

primary ** primary

Term. A term is a factor of one of the forms:

term/factor
or
term*term

Signed Term. A term immediately preceded by a + or - sign.

Simple Expression. A term or two simple arithmetic expressions separated by a + or - sign.

Arithmetic Expression. A simple expression or a signed term or either of the preceding, immediately followed by a + or - sign, immediately followed by a simple expression.

A primary of any type may be exponentiated by an integer primary and the resulting factor is of the same type as that of the element being exponentiated. A real primary may be exponentiated by a real primary, and the resulting factor is of type real. These are the only cases for which use of the exponentiation operator is defined. Figure 4-1 gives the valid combinations for exponentiation.

By use of the arithmetic operators other than exponentiation any admissible element may be combined with another admissible element of the same type.

A part of an expression is evaluated only if it is necessary to establish the value of the expression. The rules for formation of expressions imply the binding strength of the operators. The range of the subtraction operator is the term of the operator that immediately succeeds it. The evaluation may proceed according to any valid formation sequence. Use of an array element name requires the evaluation of its subscript. The type of the expression in which a function reference or subscript appears does not affect, nor is it affected by the evaluation of the actual arguments of subscript. An element whose value is not mathematically defined cannot be evaluated.

The following rules represent the derivation of all permissible expressions:

A variable, constant or function standing alone is an expression.

A(1)   JOBNO   217   17.26   SQRT(A+B)

If E is an expression whose first character is not an operator, then +E and -E are expressions.

-A(1)   +JOBNO   -217   +17.26   -SQRT(A+B)

If E is an expression then (E) is an expression meaning the quantity E taken as a unit.

(-A)   -(+JOBNO)   -(X+Y)   (A-SQRT(A+B))

If E is an expression whose first character is not an operator, and F is any expression, then: F+E, F-E, F*E, F/E and F**E are all expressions.

-(B(I, J)+SQRT(A+B(K, L)))   1.7E-2**(X+5.0)
-(B (I+e, 3*J+K) +A)

	**	Real	Integer
Base	Real	Valid	Valid
	Integer	Invalid	Valid

Figure 4-1. Exponent

The mode of an expression may be either integer or real, and is determined by the modes of its elements, which must be the same with the following exceptions:

A real quantity can appear in an integer expression only as an argument of a function.

I+LFUNC (B)

An integer quantity can appear in a real expression only as an argument of a function, as a subscript, or as an exponent.

AFUNC (I+2)   A(I, J+1)   B**N

The order of evaluation of expressions is established by the use of parentheses in the statement. If parentheses are not indicated, the following conventions of mathematics apply:

The hierarchy of operations, in order of precedence is: exponentiation, followed by multiplication and division, followed by addition and subtraction.

Within the same hierarchy of operations, evaluation proceeds from left to right.

Examples:

X+Y*Z	is interpreted as	X+(Y*Z)
W*X/Y*Z	is interpreted as	((W*X)/Y)*Z
B**2-4.*A*C	is interpreted as	(B**2)-((4.*A(*C))
X-Y-Z	is interpreted as	(X-Y)-Z
X/Y/Z	is interpreted as	(X/Y)/Z
-X**3	is interpreted as	-(X**3)

4.2 ARITHMETIC ASSIGNMENTS AND REPLACEMENTS

The assignment statement is used to replace the value of a variable with the results of the evaluation of an expression.

Form: v = e, where v is any variable or array element name, and e is an arithmetic expression.

If the mode of the expression is different than the mode of the variable, the value of the expression will be converted to cause its mode to be compatible with the mode of the variable. Figure 4-2 defines the rules for assignment of e to v.

v	e	ASSIGNMENT RULE
Real	Real	Assign
Real	Integer	Float and Assign
Integer	Integer	Assign
Integer	Real	Fix and Assign

Figure 4-2

SECTION V
CONTROL STATEMENTS

5.1 GENERAL

Each statement in a FORTRAN program is executed in the order of its appearance in the source program, unless this sequence is interrupted or modified by a control statement. This section of the manual describes the various control statements used in DATA 620 Series FORTRAN.

5.2 GO TO STATEMENTS

GO TO statements transfer logical control from one section of a program to another. Basic FORTRAN includes two forms of the GO TO statement; unconditional and computed.

5.2.1 Unconditional GO TO

An Unconditional GO TO is of the form: GO TO k, where k is a statement label reference.

Execution of this statement causes the statement identified by the label k to be executed next in sequence.

Example:

```
      GO TO 72
        .
   71 V7 = HQ (5) + Y**L
        .
        .
        .
   72 V7 = HQ (4) + X**J
```

In this example, execution of the GO TO 72 statement causes statement number 71 and any succeeding statements to be by-passed. Execution is resumed with statement number 72.

5.2.2 Computed GO TO

The computed GO TO statement is of the form: GO TO (k_1, k_2, ..., k_n), i, where the k's are statement label references, and i is an integer variable reference.

Execution of this statement causes the statement identified by the statement label k_j to be executed next in sequence where j is the value of i at execution time. Valid execution of this statement is dependent upon the value of the integer variable such that 1 is less than or equal to j, and j is less than or equal to n.

Example:

```
GO TO (98,405.3), n
```

Execution of the statement in the example will cause control to be transferred to the statement labeled 98,405 or 3 if the value of the variable integer n is 1, 2 or 3 respectively. If n contains an integer other than 1, 2 or 3, the results of the transfer cannot be predicted.

5.3 ARITHMETIC IF STATEMENT

It is often necessary to alter the logical flow of a program on the basis of the results of an arithmetic test. The IF statement is a conditional transfer that will execute this level of control, and is of the form:

IF (e) k_1, k_2, k_3

The arithmetic IF is a three-way transfer. Execution of this statement causes the expression (e) to be evaluated, following which, the statement identified by the label k_1, k_2, k_3 is executed next in sequence, as the value of (e) is less than zero, equal to zero, or greater than zero, respectively.

Example:

```
IF (1) 10, 11, 12
10 V7 = HQ (5) + Y**L

GO TO 13
11 V7 = HQ (4) + X**J

GO TO 13
12 V7 = HQ (3) + X**L

13 Next Statement
```

In this example, execution of the IF (1) 10, 11, 12 statement causes one of the following actions: for a negative value of I, statement number 10 is executed in sequence; for a zero value of I, statement number 10 and any succeeding statements are by-passed and statement number 11 is executed; for a positive, non-zero value of I,

statements 10 through 11 and any statement following statement 11 are by-passed, and statement number 12 is executed.

5.4 CALL STATEMENT

The CALL statement causes a transfer of execution control to a subroutine type subprogram, and is of one of the forms: CALL s (a_1, a_2, ..., a_n) and CALL s, where s is the name of a subroutine and the a's are actual arguments that will replace the dummy arguments in the called subroutine. Arguments may be variable names, array element names, array names, or any other expression. They must, however, be indicated in order, number and type with the corresponding dummy arguments of the subroutine.

Execution of the call statement transfers control to the designated subroutine. The arguments declared in the statement line are associated with the dummy arguments that are parameters of the executable statements of the subroutine. Control is then passed to the first executable statement of the called subroutine. Control will be returned to the first executable statement following the CALL statement upon execution of the RETURN statement in the subroutine. Examples of calling sequences to subroutines are shown below.

```
CALL TEST (A, I)
CALL EXIT
```

The first example will transfer execution control to the subroutine labelled TEST, and the inclusion of the parameters or arguments A and I in the subroutine. The second example will cause execution control to be transferred to the subroutine labelled EXIT. Any arguments required for execution of EXIT are self contained in the logic of the subroutine.

5.5 RETURN STATEMENT

The execution of a RETURN statement results in the exit from a subprogram, and is expressed in the form: RETURN.

A RETURN statement defines the logical end of a procedure subprogram, and therefore may appear only in a subprogram. Execution of the statement returns logical control to the current calling program unit. Each subprogram must contain at least one RETURN statement.

In the case of a subroutine subprogram, control is returned to the first statement immediately following the CALL statement that released control to the subroutine. In the case of a function subprogram, control is returned (with the value of the function available), to the statement that called the function subprogram.

5.6 CONTINUE STATEMENT

Form: CONTINUE.

The CONTINUE statement results in no action in an execution sequence, and therefore the statement has no effect upon the program. This statement serves as a program unit reference point.

Example:

```
IF (I) 10, 11, 12
10 V7 = HQ (5) + Y**L
     .
     .
GO TO 13
11 V7 = HQ (4) + X**J
     .
     .
GO TO 13
12 V7 = HQ (3) + X**L
     .
     .
13 CONTINUE
```

5.7 PAUSE STATEMENT

Form: PAUSE n or PAUSE, where n is an octal digit string of length from 1 to 4.

A PAUSE statement causes a temporary cessation of program execution, and displays PAUSE n (see section 8 – for display format). The statement permits operator intervention for setup or control functions, such as changing data tapes. The computer executes a Halt instruction delaying further execution until the operator selects the console Run button. Execution will resume at the first executable statement following the PAUSE statement.

Example:

```
PAUSE 01
```

5.8 STOP STATEMENT

Form: STOP n or STOP, where n is an octal digit string of length from 1 to 4.

A STOP statement causes termination of program execution, and displays STOP n (see section 8 – for display format). The program then terminates with a Halt instruction.

Example:

```
STOP 0721
```

5.9 DO STATEMENT

The DO statement is used to control repetitive execution of a group of statements. The number of repetitions is dependent upon the value of a control variable. The statement assumes one of the forms: DO n i = m_1, m_2, m_3 and DO n i = m_1, m_2, where:

> n is the statement label of an executable statement. This statement, called the terminal statement of the associated DO must physically follow and be in the same program unit as the DO statement. The terminal statement may not be a GO TO of any form, arithmetic IF, RETURN, STOP, PAUSE or another DO statement.
>
> i is an integer variable name, identified as the control variable.
>
> m_1, identified as the initial parameter; m_2, as the terminal parameter; and m_3, as the incrementation parameter; are each either an integer constant or integer variable reference. If the second form of the DO statement is used, a value of 1 is implied for the incrementation parameter, when the DO statement is executed, the values of m_1, m_2, and m_3 must be greater than zero.

Associated with each DO statement is a range that is defined to be those executable statements from and including the first executable statement following the DO, to and including the terminal statement defined by the DO. A special situation occurs when the range of a DO contains another DO statement. In this case, the range of the contained DO must be a subset of the range of the containing DO.

The control variable is assigned the value represented by the initial parameter. This value must be less than or equal to the value represented by the terminal parameter.

The range of the DO is executed.

If control reaches the terminal statement, and after execution of the terminal statement, the control variable of the most recently executed DO statement associated with the terminal statement is incremented by the value represented by the associated incrementation parameter.

If the value of the control variable after incrementation is less than or equal to the value represented by the associated terminal parameter, the action is repeated with the understanding that the range in question is that of the DO, the control variable of which was most recently executed.

If the value of the control variable is greater than the value represented by its associated terminal parameter, the DO is said to be satisfied, and the control variable becomes undefined.

If there were one or more other DO statements referring to the terminal statements in question, the control variable of the next most recently executed DO statement is incremented by the value represented by the associated incrementation parameter until all DO statements referring to the particular termination statement are satisfied, at which time the first executable statement following the terminal statement is executed.

Upon exiting from the range of a DO by execution of a GO TO statement or an arithmetic IF statement, that is other than by satisfying the DO, the control variable of the DO is defined and is equal to the most recent value attained.

A GO TO statement or an arithmetic IF statement may not cause control to pass into the range of a DO from outside its range. When a procedure reference occurs in the range of a DO, the actions of that procedure are considered to be temporarily within that range, i.e., during the execution of that reference.

The control variable, initial parameter, terminal parameter and incrementation parameters of a DO may not be redefined during the execution of the range of that DO.

If a statement is the terminal statement of more than one DO statement, the label of that terminal statement may not be used in any GO TO or arithmetic IF statement that occurs anywhere but in the range of the most deeply contained DO with that terminal statement.

Example:

```
DO 607 K1 = 2, ID, 3
```

The foregoing statement would cause K1, the control variable, to be set to the value of the initial parameter, 2. Execution would proceed at the statement immediately following, down to and including the statement identified by the label 607. After each execution of the loop, K1 is incremented by the incrementation parameter, 3, and evaluated in relation to the current value of the terminal parameter, I.D. If the current value of ID is greater than K1, execution control is transferred to the statement following that identified by the label 607, otherwise the DO cycle is repeated.

SECTION VI
INPUT / OUTPUT STATEMENTS

6.1 GENERAL

Input statements provide a program with the means of receiving information from external sources. Output statements allow the transmission of program data to extend sources. These external sources may be devices such as magnetic tape and paper tape handlers, typewriters, and punch card processors.

There are two types of input-output statements.

(1) READ and WRITE statements
(2) Auxiliary statements

The first type cause the transfer of records of sequential files to and from the program. This data may be formatted information consisting of strings of characters, or unformatted information consisting of binary word values in the form in which they normally appear in storage. The second statement type consists of the BACKSPACE and REWIND statements which provide for positioning of magnetic tapes, and the ENDFILE statement which provides for closing of a file.

Input-Output statements reference input-output units and, formatted information, format specifications. An input-output unit is identified by a logical unit number, u, which may be either an integer constant or a variable name that references an integer constant. Logical unit number assignments for the DATA 620/i FORTRAN may be found in appendix L. The format specification is defined by a FORMAT statement having the statement label f. This statement must appear in the same program as the input-output statement.

6.2 INPUT-OUTPUT LISTS

The input list specifies the names of variables and array elements to which input values are assigned. The output list specifies the names of variables and array elements whose values are transmitted. Input and output lists are of the same form.

6.3 SIMPLE LISTS

Simple lists have the form: $m_1, m_2, m_3 \ldots, m_n$ where the m_i are the names of real or integer variables or array elements. The comma characters separate each individual name in the list. The period characters signify possible additional list items. List elements may be enclosed in parentheses.

Example:

INPUT LISTS	OUTPUT LISTS
A	B
C (26, L)	I (10, 10)
R, K, D, (I, J)	S, (R, K), F (1, 25)

An array variable which is not subscripted in a list is considered equivalent to the listing of each successive element of the array. If B is an array, the list B is equivalent to B (1, 1), B (2,1), B (3, 1), . . . , B (1, 2), B (2,2), . . . , B (j, k) where j and k are the subscript limits of B.

6.4 DO-IMPLIED LISTS

A DO-implied list is a simple list followed by a comma character and an expression of the form: $i = m_1, m_2, m_3$ or $i = m_1, m_2$.

The elements i, m_1, m_2, and m_3 have the same meaning as defined for the DO statement. The DO implication applies to all simple list items enclosed in parentheses with the implication. For input lists, i, m_1, m_2, and m_3 may appear within this range only as subscripts.

Examples:

(X (I), I = 1, 4)	X (1), X (2), X (3), X (4)
(Q (J), R (J), J = 1, 2)	Q (1), R (1), Q (2), R (2)
(G (K), K = 1, 7, 3)	G (1), G (4), G (7)
((A (I, J), I = 3, 5), J = 1, 2)	A (3, 1), A (4, 1), A (5, 1), A (3, 2), A (4, 2), A (5,2)
(X (K), K = 1, 2), I, (R (J), J = 3, 5)	X (1), X (2), I, R (3), R (4), R (5)

6.5 READ STATEMENTS

These statements are used to obtain data values from an external source. The data values are input in either formatted or unformatted mode. The form of a formatted READ statement is: READ (u, f) k.

The verb READ and the parentheses must appear in this form.

Execution of this statement causes information to be transmitted from the external source whose logical unit number is defined by u. This data is scanned and converted as specified by the format specification, f, and the resulting values are assigned to the variable names defined in the list, k.

The form of an unformatted READ statement is: READ (u) k.

The verb READ and the parentheses must appear in this form. This statement causes data to be input in binary form from the unit defined by u. The values are assigned to the variable names defined in the list, k.

Examples:

```
READ     (1, 44) A, B, C
READ     (2) R, S
READ     (N, 12) A, (R (I), I = 1, 10)
READ     (L) S, (T (J), J = 1, N)
```

All information appearing on external sources is divided into records. Each time a READ statement is executed a new record is processed. The number of records input by a single READ statement is determined by the list and format specification. If only part of a record is input the remainder of the record is lost as the next READ processes the next record. Records are read sequentially until the list is exhausted. Only enough values are read to fill the list.

The list, k, in an unformatted read statement may be left blank to skip a record.

The record size for formatted data is 80 characters except when the device is the Teletype keyboard or paper tape in which case the record size is variable with a maximum of 80 characters processed per record. Unformatted records are 64 binary words in length.

6.6 WRITE STATEMENTS

WRITE statements are used for the purpose of transferring program data to external devices. This data may be formatted or unformatted. The form of a formatted WRITE statement is: WRITE (u, f) k.

The verb WRITE and the parentheses must appear in this form.

Execution of this statement causes records to be written on the device referenced by u. The contents of the records are the values taken sequentially from the list k converted according to the format specification f.

The form of an unformatted WRITE statement is: WRITE (u) k.

The verb WRITE and the parentheses must appear in this form.

Execution of this statement causes binary information from the list k to be written in records on the unit defined by u.

Example:

```
WRITE       (1, 5) A, B, C
WRITE       (7) R, S, T
WRITE       (K, 12) X, (Y (J), J = 1, M), I
WRITE       (N) W, Z, (F (K), K = 1, 5)
```

Several record may be written with a single WRITE statement. The number of records is determined by the list and the format specifications. Successive records are written until the data is exhausted. If the data does not fill a record, the record is filled with blanks.

6.7 REWIND STATEMENT

This statement is of the form: REWIND u.

Execution of this statement cause the magnetic tape unit defined by u to be rewound. If u is not a magnetic tape, no action is taken.

6.8 BACKSPACE STATEMENT

This statement has the form: BACKSPACE u.

The BACKSPACE statement causes the magnetic tape unit defined by u to be backspaced one record. If u is not a magnetic tape, no action is taken.

6.9 ENDFILE STATEMENT

This statement has the form: ENDFILE u.

When this statement is executed, a file mark is written on the magnetic tape defined by u. No action is taken if u is not a magnetic tape.

6.10 FORMAT STATEMENTS

FORMAT statements are used with input-output operations to specify conversion and editing of information between program storage and external representation. FORMAT statements are non-executable and must have a statement label to be referenced by input-output statements. Conversion performed according to a FORMAT statement during output is in general the reverse of conversion performed during an input operation.

A FORMAT Statement is expressed as: n FORMAT (f_1, f_2, f_3, . . . , f_n), where n is the statement label and the f_i are field specifications. The noun FORMAT and the parentheses must appear in this form. The comma characters are required only when

ambiguities would arise from not separating field specifications. The period characters signify possible additional field specifications and would not actually be present.

6.11 FIELD SPECIFICATIONS

Field specifications describe the type of conversion and editing to be performed on each variable appearing in the input-output list. Field specifications may be any of the following forms:

rFw.d
rEw.d
rIw
nHs
nX

where:

1. The characters F, E, and I indicate the manner of conversion for variables in the list.
2. The characters H and X represent character data to be input-output directly from the format.
3. The character / represents the end of a record.
4. w and u are non-zero integer constants defining the width of the field (including digits, decimal points, algebraic signs) in the external character string.
5. d is an integer specifying the number of fractional digits appearing in the external string.
6. r is an optional, non-zero integer indicating that the specification is to be repeated r times.
7. s is a string of acceptable FORTRAN characters.

6.12 F CONVERSION

Form: rFw.d

Only real data may be processed by this form of conversion.

Output. The field is right justified with as many leading blanks as necessary to fill w. Negative values are preceded by a minus sign. Internal values are converted to fixed point decimal numbers and rounded to d decimal places.

For a field specification of F10.4:

368.4	is converted to	368.4000
12.0	is converted to	12.0000
-17.90767	is converted to	-17.9077
37.5E-2	is converted to	0.3750

If a value requires more positions than allowed by w the most significant digits, including sign if negative, are output. The error indication is designated by an asterisk in the least significant character position.

For a field specification of F6.4:

4739.76	is converted to	4740.0*
-12.463	is converted to	-12.5*

Input. Input strings are decimal numbers of length w with d characters in the fractional portion. Blanks are treated as zeros. If a decimal point is present in a value the fractional portion of the value is explicitly defined by that decimal point character. A comma (,) terminator may be used to override the w specification. Terminated fields are treated as normal fields with leading zeros. A comma alone defines a zero value for the field.

For a field specification F8.3:

35	is converted to	0.035
964372	is converted to	964.372
0.53821	is converted to	0.53821
-16.402	is converted to	-16.402
-12	is converted to	-0.012
47.E-4	is converted to	0.0047
36,	is converted to	0.036
-0.75,	is converted to	-0.75
,	is converted to	0.0

6.13 E CONVERSION

Form: rEw.d.

Only real data may be processed by this form of conversion.

Output. Internal values are converted to decimal values of the forms: .ddd...dE ee and .ddd...dE-ee, where ddd...d represent d digits, while ee is a decimal exponent. The leading decimal point and E characters are present exactly as shown. Internal values are rounded to d digits and negative values are preceded by a minus sign. The external field is right justified and preceded by blanks to fill the width, w. This field width includes the exponent digits, the sign of the exponent (minus or space), the letter E, the magnitude digits, the decimal point, and the sign of the value (minus or space). This means that the field width should correspond to the relation: w ≥d + 6.

If w is less than (d + 6) the format is in error.

For the field specification E12.5:

76.573	is converted to	0.76573E 02
58796.341	is converted to	0.58795E 05
-369.7583	is converted to	-0.36976E 03
0.006873	is converted to	0.68730E-02
0.2	is converted to	0.20000E 00
-0.0000054	is converted to	-0.54000E-05

Each external value is of field width w with d characters in the fractional part of the value. The value is right justified with all blanks counting as zeros. A minus sign may be placed preceding the value of the exponent. A decimal point placed in the fractional part takes precedence over the d specification. The character E should be present to separate the value and the exponent. If not, the exponent is taken as the two least significant digits. A comma (m) terminator may be used to override the w specification. Terminated fields are treated as normal fields with leading zeros. A comma alone defines a zero value for the field.

For a field specification E10.3:

123E3,	is converted to	123.0
12874E2	is converted to	1287.4
-563E-02	is converted to	-0.00563
-6.7563E05	is converted to	-675630.0
398E00	is converted to	0.398
5387601	is converted to	538.76
5455-01	is converted to	0.5455

6.14 I CONVERSION

Form: rIw

Only integer data may be processed by this form of conversion.

Output. Internal values are converted to integer constants. Negative values are preceded by a minus sign. Each field is right justified and filled with leading blanks.

For the field specification I6:

281	is converted to	281
-43567	is converted to	-43567

If the data requires more character positions than allowed by the width w, only the least significant w positions are output.

For the field specification I2:

281	is converted to	81
-6374	is converted to	74

Input. External input values are right justified with the width w. Blanks are counted as zeros. Input values must be integer values. A preceding minus sign may be placed on a value. A comma (,) terminator may be used to override the Iw specification. Terminated fields are treated as normal fields with leading zeros. A comma alone defines a zero value for the field.

For the field specification I4:

120	is converted to	120
-144	is converted to	-144
1 2	is converted to	1020
-3,	is converted to	-3

6.15 H CONVERSION

In DATA 620/i FORTRAN, Hollerith information consists of the legal FORTRAN character set plus the additional characters $, !, ", #, %, &, ', :, ;. Information input from the typewriter or paper tape is converted to an internal code used by FORTRAN. When this information is output the internal codes are converted to the appropriate typewriter or paper tape codes.

Form: wHs.

Output. The number of characters, w, in the string, s, should contain exactly the number of characters specified so that characters from other fields are not taken as part of the string.

Blanks are counted as characters in the string.

Examples:

SPECIFICATION	EXTERNAL OUTPUT
1HR	R
8H STRING	STRING
12HX (1, 3) = 12.0	X (1, 3) = 12.0

Input. The w characters in the string s are replaced by the next w characters from the input record. The resultant is a new string in the field specification.

For Example:

SPECIFICATION	INPUT STRING	RESULTANT SPECIFICATION
5H12345	ABCDE	5HABCDE
7H TRUE	FALSE	7HFALSE
8H	MATRIX	8HMATRIX

This feature can be used to change titles, dates, headings, etc., which are output with the program data.

6.16 X SPECIFICATION

Form: wX.

This specification causes no conversion to occur. On output, w blanks are inserted in the external record. On input, w spaces are skipped from the input record.

Example of output:

SPECIFICATION	OUTPUT
1HA, 4X, 2HBC	A BC
4X, 3HABC	ABC
1X, 3HABC, 3X	ABC

Example of input:

SPECIFICATION	INPUT STRING	RESULTANT INPUT
F4.1, 3X, F3.0	12.5RRR120	12.5, 120.

The RRR characters are ignored by the 3X specification.

6.17 / SPECIFICATION

Form: /.

Each slash (/) specified in the format causes the termination of a record and processing of the next record. Successive slashes (///...//) cause successive records to be ignored on input, and successive blank records to be written on output. A slash separating two field specifications removes the need for a comma separator.

For example:

F5.4,/4F10.3 is equivalent to F5.4/4F10.3

Output Example:

For a specification (1HA/1HB/1HC/1HD) the resultant output records are:

A
B
C
D

Input Example:

Using the four records output from the previous example, an input specification (1H1/1H2//1H3) produces the resultant specification (1HA/1HB//1HD).

6.18 REPEAT SPECIFICATIONS

The F, E, and I field specifications may be repeated by using the repeat count r in the forms rFw.d, rEw.d, and rIw.

Examples:

```
   4F10.5,F3.6  is equivalent to  F10.5,F10.5,F10.5,F10.5,F3.6
    2F4.1,2E7.1  is equivalent to  F4.1,F4.1,E7.1,E7.1
2F5.2,3I6,2E8.2  is equivalent to  F5.2,F5.2,I6,I6,I6,E8.2,E8.2
```

Repetition of a group of field specifications is accomplished by enclosing the group in parentheses preceded by an integer repeat count. If no repeat count is specified the count is taken as one.

Examples:

```
   2(F10.5, I6)  is equivalent to  F10.5,I6,F10.5,I6
 2(E9.3,F7.1/I4)  is equivalent to  E9.3,F7.1/I4,E9.3,F7.1/I4
3(4F5.0,2E8.2)  is equivalent to  4F5.0,2E8.2,4F5.0,2E8.2,4F5.0,2E8.2
```

Example:

```
50 FORMAT          (4X,2(I5,6F8.2)/3(E12.7,F6.4),2I4)
```

6.19 FORMAT CONTROL AND LIST INTERACTION

Execution of a formatted READ or WRITE statement initiates format control. The conversion performed on data depends on information jointly provided by the next element of the input-output list and the next field specification of the FORMAT statement. If there is a list, at least one field specification of type E, F, or I should be present in the FORMAT statement.

Execution of a formatted READ statement causes one record to be input. To each E, F, or I specification there corresponds one element in the list. To each H or X specification there is no corresponding element in the list and the format control communicates information directly with the record. Whenever a slash is encountered, or the entire input record is processed, the record is terminated. If more input is necessary the next record is input. Any unprocessed characters of a record are skipped when a slash is encountered.

A READ statement is terminated upon expiration of the list if: 1. the next specification is an E, F, or I; 2. the format control has reached the last outer right parenthesis of the FORMAT statement. If the list expires and the next specification is an H or X, data is processed (with the possibility of additional records being input) until one of the above two conditions is met.

If the format control reaches the rightmost parenthesis of the FORMAT statement and more list remains to be processed the following steps are taken: 1. a new record is input and any remaining data in the previous record is ignored; 2. format control reverts to the point immediately following the last left parenthesis encountered. If group repeat specifications exist in the format, this point is at the rightmost group of the format. The repeat count is not taken into consideration. If no groups are present, the format is started from the beginning.

When a formatted WRITE statement is executed, records are written each time 120 characters or (72 characters in the case of teletype pegboard records) have been processed, a slash is encountered, or the format control terminates. The format control terminates by one of the two methods described for READ termination. Incomplete records are filled with blanks to maintain standard record lengths.

SECTION VII

PROGRAMS AND SUBPROGRAMS

7.1 GENERAL

An executable FORTRAN program consists of a main program and any required subprograms. Subprograms may be defined by the programmer or may be contained in the system library. Each program or subprogram must contain at least one executable statement.

7.2 MAIN PROGRAMS

A main program is a program unit consisting of a set of FORTRAN statements, comment lines, and an END line. The program may be preceded by specification statements. If so, these statements must be in the following order: DIMENSION, COMMON, and EQUIVALENCE.

A main program cannot contain a subprogram definition statement, namely:

a FUNCTION statement
a SUBROUTINE statement

A main program may contain calls to other subprograms or may contain statement function subprobrams.

7.3 SUBPROGRAMS

Subprograms are program units which may be called by other programs or subprograms. Subprograms are categorized as one of the following:

Statement functions
Intrinsic functions
FUNCTION subprograms
SUBROUTINE subprograms

The first three are categorized as functions and the last as subroutines.

Functions are programmed procedures which are often used to provide solutions to mathematical functions. Function references may be used in the same manner as references to variables in an expression. For example: X = AB*SIN (Y) - C*COS (Y*Z), where SIN is the name of the sine function, COS is the name of the cosine function, and (Y) and (Y*Z) are their respective argument lists. The value returned for a function reference is of the same mode as the function name, corresponding to the rules for real and integer symbolic names.

7.4 STATEMENT FUNCTIONS

A statement function is defined internally to the program unit in which it is referenced. All statement functions must precede the first executable statement and must follow any specification statements of the program unit.

A statement function is defined in a single expression of the form: $f(a_1,a_2,a_3,\ldots a_n) = e$, where f is the function name, the a_i are the arguments, and e is an expression. The resultant value of the function is either a real or integer value corresponding to the function name. The a_i are distinct variable names and are called dummy arguments. These serve to indicate the type, number, and order of the function arguments. The expression e is an arithmetic expression and may contain references to previously defined statement functions.

A statement function is referenced by a function call, $f(a_1,a_2,a_3,\ldots,a_n)$, appearing in an arithmetic expression. A statement function may only be referenced within the program unit in which it is defined. The arguments used in the reference must agree in type, number, and order with the corresponding dummy arguments.

Example:

The statement function:

```
SF (X) = A*X**2+B*X+C
```

may be referenced in the program by:

```
W = SF (Y)
```

7.5 INTRINSIC FUNCTIONS

Intrinsic functions are commonly used subprograms and are contained in the FORTRAN library. The symbolic names and meanings of the intrinsic functions are shown in figure 7-1.

An intrinsic function is referenced by a function call in an arithmetic expression. The arguments in the argument list must agree in type, number, and order with those shown in figure 7-1.

Example:

```
   IF (SIGN(W,X))  1,2,2,
1  W=ABS (X) - ABS (Y)
2  S=W*FLOAT (I*J)
   K=IFIX (X)+J
```


INTRINSIC FUNCTION	DEFINITION	NUMBER OF ARGUMENTS	SYMBOLIC NAME	TYPE OF: ARGUMENT	TYPE OF: FUNCTION
Absolute Value	a	1	ABS IABS	Real Integer	Real Integer
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of Sign	Sign of a2 times |a1|	2	SIGN ISIGN	Real Integer	Real Integer

Figure 7-1. Table of Basis Intrinsic Functions

7.6 FUNCTION SUBPROGRAMS

A function subprogram is defined externally to the program unit by which it is referenced. A function subprogram is defined by having as its first statement, other than comment lines, a statement of the form:

FUNCTION $f(a_1, a_2, a_3, \ldots, a_n)$

where f is the symbolic name of the function and the a_i are dummy arguments. Each a_i is either a variable name or an array name. The a_i define the type, number, and order of the FUNCTION arguments.

A function subprogram is executed at the first executable statement following the FUNCTION statement. Specification statements (DIMENSION, COMMON, and EQUIVALENCE) may immediately follow the FUNCTION statement. If present, these must precede any other statement, excluding comments. The symbolic names of the dummy arguments, a_i, may not appear in an EQUIVALENCE or COMMON statement.

A function subprogram must contain at least one RETURN statement and the last statement executed in a FUNCTION must be a RETURN statement. The function subprogram is ended by an END line.

The symbolic name, f, of the FUNCTION must appear as a variable name within the subprogram. The value returned for a FUNCTION is the last value assigned to this name prior to execution of a RETURN statement. The mode of the FUNCTION value, either integer or real, is determined from the function name.

The symbolic name of the function must not appear in any nonexecutable statement within the subprogram. A subprogram may not define or redefine any of its arguments nor any variable in COMMON.

Example FUNCTION:

```
      FUNCTION   XP(A,B,1)
      DIMENSION  B(10)
      XP=0.
      DO 1 J=1,10
    1 XP=(A*B(J))**1+XP
      RETURN
      END
```

A FUNCTION is executed with a function reference by a main program or another subprogram. The actual arguments in the call must correspond in type, number, and

order with the FUNCTION dummy arguments. If a dummy argument of a FUNCTION is an array name the corresponding actual argument must be an array name.

Example:

A call for the example FUNCTION shown above would be: W+XP(R,S,K) where S is an array.

7.7 BASIC EXTERNAL FUNCTIONS

Basic external FUNCTIONS are standard subprograms contained in the FORTRAN library. These are referenced in the same manner as normal FUNCTIONS. The symbolic names and meanings of the basic external FUNCTIONS are shown in figure 7-2.

7.8 SUBROUTINE SUBPROGRAMS

A subroutine subprogram is defined externally to the program unit that references it. Subroutines, unlike functions, do not have values associated with them and cannot be referenced in an expression. Subroutines are accessed by CALL statements.

A subroutine subprogram is defined by having as its first statement, other than comment lines, a statement of the form: SUBROUTINE $S(a_1, a_2, a_3 \ldots, a_n)$ or SUBROUTINE S, where S is the symbolic name of the subroutine and the a_i are the dummy arguments of the subroutine. Each a_i is either a variable name or an array name. If no arguments are passed to the subroutine the second form above is used.

The symbolic name of the subroutine must not appear in any statement in the subprogram. The symbolic names of the dummy arguments may not appear in COMMON or EQUIVALENCE statements.

A subroutine is executed at the first executable statement. Specification statements may be contained immediately following the SUBROUTINE statement and preceding any executable statement. A subroutine must have at least one RETURN statement. The last statement executed by a subroutine must be a RETURN statement.

DATA 620/i series FORTRAN includes a subroutine named 'EXIT'. When this subroutine is referenced by a CALL statement of the form:

```
      CALL EXIT
```

the statement END OF JOB will be displayed (see section 8 -- for display format), and the program terminates with a Halt instruction.

BASIC EXTERNAL FUNCTIONS	DEFINITION	NUMBER OF ARGUMENTS	SYMBOLIC NAME	TYPE OF: ARGUMENT	TYPE OF: FUNCTION
Exponential	e^a	1	EXP	Real	Real
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real
Trigonometric sine	sine (a)	1	SIN	Real	Real
Trigonometric cosine	cos (a)	1	COS	Real	Real
Hyperbolic tangent	Tanh (a)	1	TANH	Real	Real
Square root	(a)1/2	1	SQRT	Real	Real
Arctangent	arctan (a)	1	ATAN	Real	Real

Figure 7-2. Table of Basic External Functions

Example SUBROUTINE:

```
      SUBROUTINE R(A,I,Z)
      DIMENSION A (10)
      Z=0
      DO 1 J=1, 10
    1 Z=Z+A(J)**1
      RETURN
      END
```

A subroutine is referenced with a CALL statement. The argument list in the reference must agree in type, number, and order with the dummy arguments of the subroutine. If a dummy argument is an array name, the corresponding actual argument must be an array name.

Example:

A call for the example SUBROUTINE above would be: CALL R (T,K,D) where T is an array.

7.9 DUMMY ARGUMENTS

Dummy arguments provide a means of passing information between a subprogram and the program or subprogram which called it. Both function and subroutine subprograms may have dummy arguments. A subroutine need not have any, while a function must have at least one. Dummies provide definitions of the data type, number, and sequence of subprogram parameters.

A dummy may be classified within a subprogram as a variable or an array. The actual arguments defined by a calling program or subprogram to which a dummy may correspond are: variables, array elements, arrays, expressions.

Within a subprogram a dummy may be used in much the same way as any other variable or array. A dummy man not appear in a COMMON or EQUIVALENCE statement.

The actual arguments used in a calling statement must agree in data type with the corresponding dummy arguments, that is - reals to reals, intergers to integers, and arrays to arrays. If an actual argument is an expression, the result of the expression should correspond in data type to the dummy.

A dummy array is defined to be an argument which appears in DIMENSION statement in the subprogram. A dummy array does not occupy any storage but tells the subprogram that the argument supplied in the calling statement defines the first element of an actual array. The calling argument need not have the same dimensions as the dummy array. Useful operations can sometimes be performed by defining different dimensions for the dummy and calling arguments.

Example:

```
DIMSION   A(10,10)          SUBROUTINE   FM(B)
CALL   FM(A(6,1))           DIMENSION    B(50)
```

For this case the 1 - dimensional dummy array B corresponds to the last half of the 2 - dimensional array A. If the calling statement were: CALL FM(A).

The dummy array B would correspond to the first half of the array A.

SECTION VIII

FORTRAN OPERATING INSTRUCTIONS

8.1 GENERAL

The DATA 620/i basis FORTRAN system operates in a minimum configuration of 8192 words of memory and an ASR-33/35 teletype. FORTRAN programs and subprograms are compiled by the basic FORTRAN compiler. FORTRAN compatible machine language subprograms are assembled by the DAS assembler version I, mod F. The FORTRAN loader loads main programs and all required subprograms into memory for execution. The FORTRAN run-time library provides input/output, control, and mathematical functions required at execution time.

8.2 COMPILER OPERATING INSTRUCTIONS

The DATA 620/i basic FORTRAN compiler translates FORTRAN source programs to relocatable machine language programs in a single pass. FORTRAN statements may be input from the teletype keyboard or paper tape reader, the card reader, the high speed paper tape reader or magnetic tape. Object code is output via the teletype or high speed paper tape punch or magnetic tape. Error diagnostics, source listings and object listings are provided on the teletype or line printer. Input/output and listing options are selected at the teletype keyboard for each program to be compiled.

8.3 PRELIMINARY OPERATIONS

The DATA 620/i basic FORTRAN compiler is supplied as an absolute binary object tape. The compiler is loaded into memory by the standard binary loader and occupies the first 13500 (8) words of memory. (See programming reference manual for procedure to load absolute object programs.) Entry to the compiler is at location 0. Upon entry, the compiler will execute a HALT 0777 with the A register set to the upper limit of compiler used memory (15777 standard). This limit may be modified by resetting the A register. (See appendix M for compile time memory map.) To compile programs press RUN.

8.4 NORMAL OPERATIONS

For each program to be compiled a ?= will be typed on the teletype printer requesting input/output selection. The operator should respond by typing one of the following characters to indicate the input device: C (card reader), K (teletype keyboard), P (paper tape), 0 through 3 (magnetic tape, units 0 through 3); followed by one of the following characters to indicate the output device: C (card punch), P (paper tape), 0 through 3 (magnetic tape, units 0 through 3); followed by an (optional) listing selection character: S (source listing), Ø (object listing), B (both source and object listings); followed by the character >, followed by the (optional) 1 to 6 character program name, followed by @ for a carriage return and line feed.

Example:

? = CPS > MATRIX @

> C for input cards, P for output paper tape, S for list source with program name MATRIX. Following input/output selection, source statements are read and object records are output through the selected devices. Error diagnostics and selected list options are printed on the teletype or line printer (if available). Upon detecting and END statement (followed by a non-bland statement), the compiler will produce a program map listing all variables, constants (in octal), and required subprograms. Having listed the program map the compiler will type a ? = to permit compiling another program.

8.5 INPUT RECORDS

Input to the compiler is a series of FORTRAN statements each of which appear in one or more input records. Records may be fixed or variable in length depending on the device, however, only the first 72 characters of each record are used by the compiler. Any illegal characters are treated as blanks. Blank records are ignored. END statements must be followed by at least one non-blank record (another END statement is suggested).

Keyboard and paper tape records are variable length and are terminated by a carriage return and line feed in that order. The character > may be used to TAB to column 7, and the character ← may be used to clear the input buffer and reset to column 1. For keyboard input the teletype bell is rung to notify the operator that source input is required.

Card records are a fixed length of 80 characters. The special characters > and ← are treated as blanks.

Magnetic tape records are a fixed length of 84 characters, and should be card or paper tape images with blank padding characters. The special characters > and ← are permitted as defined for paper tape. Carriage return and line feed characters are permitted but ignored.

8.6 OUTPUT RECORDS

Object records are a fixed length of 64 words and are output from time to time as they are created. Paper tape object programs are punched with leader and trailer records.

Magnetic tape object programs are terminated by an end of file. All main programs are terminated by an end-of-tape record. Refer to appendix N for object record format.

All error diagnostics are of the form: ERR xx a . . . a, where xx is a number from 1 to 15 (notification error) or T followed by a number from 1 to 9 (terminating error), and a . . . a represents the last (up to 16) characters encountered in the statement being processed. The right most character indicates the point where the error was discovered (the character @ indicates end of statement). If a terminating error is discovered object output is terminated, but source code is continued to detect any further errors.

8.7 NOTIFICATION ERRORS

1. Construction
2. Usage
3. Mode
4. Illegal DO Termination
5. Improper Statement Number
6. Common Base Lowered
7. Illegal Equivalence Group
8. Reference to Non-Executable Statement
9. No Path to this Statement
10. Multiply Defined Statement Number
11. Invalid Format Construction
12. Spelling Error
13. Format with No Statement Number
14. Function Not Used as Variable
15. Truncated Value

8.8 TERMINATING ERRORS

T1. Construction
T2. Usage
T3. Data Pool Full
T4. Illegal Statement
T5. Improper Use of Name
T6. Improper Statement Number
T7. Mode
T8. Constant Too Large
T9. Improper DO Nesting

8.9 OPTIONAL LISTINGS

Source and object records may be listed if desired. Source records are listed as they are input. Object records are listed from time to time as they are created. Each object record consists of a varying number of 2 and 4 word data/instruction entries. The object record listing consists of one line for each entry. Two word entries are of the form abbc vvvvvv, and four word entries are of the from abbc nnnnnn vvvvvv, where a is the control code, bb is the sub code, c is the pointer number, nnnnnn is a 1 to 6 letter subprogram name and vvvvvv is a 6 digit octal value or instruction. See appendix N for object record format and codes.

8.10 PROGRAM MAP

Upon processing the END statement the compiler will list the program map. The first three lines of the map define the size of the program, data and common areas and are of the form a, *SIZE mmmmmm; where a is the area (0 = program, 1 = data, 2 = common), and mmmmmm is the octal size. For programs with no terminating errors the following information is also listed.

a)	a,	11111	nnnnn	Variable
b)	a,	11111	cccccc cccccc	Constant
c)	S,	11111	nnnnn	External Subprogram
d)	#,	11111	ssss	Statement Number
e)	X,	11111	ssss	Undefined Statement Number

Where a is the area, 11111 is the relative location of the item or the last reference to the subprogram or statement number, cccccc cccccc is a two word octal constant, and ssss is a statement number.

8.11 FORTRAN LOADER OPERATING INSTRUCTIONS

The FORTRAN loader is designed to operate in a DATA/i 620 computer with at least 8192 words of memory. Its function is to load relocatable object programs produced by the DATA 620/i FORTRAN compiler and FORTRAN compatible subprograms produced by the DATA 620/i assembler. Object program input is from either paper or magnetic tape and is selected from the teletype keyboard. Load maps and error diagnostics appear on the teletype printer. See appendix M for load time memory map.

8.12 PRELIMINARY OPERATIONS

The FORTRAN loader is supplied as an absolute binary object tape and is loaded into memory by the binary loader (see programming reference manual, for loading procedure). The FORTRAN loader occupies locations 000 through 077 and 014000 through 015740.

(Locations 0100 through 0277 are reserved for loader generated pointers.) The first program to be loaded must be a FORTRAN compiled main program. Prepare the input unit, clear the registers and RUN at location STRT (014140). The message IN will appear on the teletype, requesting input selection. To select paper tape input type P. To select magnetic tape, type the unit number (0, 1, 2, or 3). If the selected unit is attached and ready the loader will load the main program (at location 0300). The teletype will then type RQ followed by a list of the subroutines required, followed by another input selection request.

8.13 LOADING SUBPROGRAMS

To effect the most efficient use of the loader, it is recommended that subprograms be loaded in the following order:

a. Customer produced subprograms.
b. FORTRAN input/output subprograms.
c. FORTRAN math subprograms.
d. FORTRAN utility subprograms.

Prepare and select the input unit as for main programs. The loader will load all required subprograms until an end of tape record is detected, at which time the list of required subprograms is generated and input selection is again requested. (NOTE: The end of tape record is not produced for subprograms by either the compiler or the assembler, but is supplied as a separate tape labeled FORTRAN END-OF-TAPE, and should be spliced on the end of the users subprogram library tapes. Standard FORTRAN library tapes are delivered with end-of-tape records.)

If two or more subprograms have the same name, only the first such subprogram input will be loaded. When all required subprograms have been loaded, the message GØ will be typed followed by the load map, which lists each subprogram loaded and its entry point. To execute the loaded program press RUN. The load map may be forced by running at location RUN (015025). Execution of the main program may be forced by running at location 0300.

8.14 ERROR DIAGNOSTICS

An error in the loading process will cause type out of an error message and the load map. A minus sign will precede the address of each subprogram which has not been loaded, in this case, the address represents the last location at which the subprogram is requested. All errors except checksum errors are non-recoverable. The error must be corrected and the loading process re-initialized.

CK Checksum error. Backspace the input tape one record and press RUN for another attempt.

AR Area reference. An attempt has been made to load a value to an area not yet defined.

CE Compiler error. A terminating error occurred at compile time.

CS Common size. A secondary use of blank common has occurred that is larger than the initially defined area.

SZ Program size. The program being loaded is too large for the memory available.

8.15 EXECUTION OF FORTRAN PROGRAMS

All FORTRAN main programs are loaded and entered at location 300(8). Required subprograms are loaded as they are input in successive blocks of memory. Common storage normally overlays the FORTRAN loader, which leaves the AID II routines and absolute binary loader in memory at their standard locations. Locations 0 through 77(8) are unused and locations 100(8) through 277(8) contain program and data pointers used by the program and subprograms to be executed.

To execute a FORTRAN program initialize the input/output devices selected, clear the console registers, set the program counter to 300(8), press SYSTEM RESET and RUN.

8.16 PROGRAMMED HALTS

DATA 620/i FORTRAN provides fo 3 types of programmed halts: STOP, PAUSE, and EXIT. STOP causes the program to execute a HALT 0777 with the stop number displayed in 4 bit BCD in the A register and the B register set to -1. A STOP implies end of job. PAUSE causes the program to execute a HALT 0000 with the pause number displayed in 4-bit bcd in the A register and the B register set to 0. The program may be continued by pressing SYSTEM RESET and RUN. EXIT causes the program to execute a HALT 0777 with the A and B registers set to -1 and signifies end of job. All programmed halts display error bits in the X register.

8-17 ERROR BIT DESIGNATIONS

Bit 0 indicates floating point overflow.
Bit 1 indicates divide check.
Bit 2 indicates fixed point overflow.
Bit 3 indicates indeterminate function.
Bit 4 indicates a log error.
Bit 5 indicates square root error.
Bit 6 indicates GO TO error.

8-18. ERROR HALTS

The following error halts are generated by the run time input/output package. These errors cause a 4 character message to be typed on the teletype printer followed by a call to EXIT.

FRMT:	Format error.
MODE:	Data mode error (floating point vs. integer).
DATA:	Input data field error.
UNIT:	Unit not attached or not available.
TAPE:	Checksum or tape parity error.

8.19 BINARY INPUT/OUTPUT

All binary input/output records are a fixed length of 64 words. Paper tape records are punched with a record mark and checksum (see appendix N -- for a detailed format). Checksum errors encountered on input will cause a TAPE error.

8.20 BCD INPUT/OUTPUT

Bcd records may be fixed or variable in length depending on the device, however, only the first 80 characters are processed.

Keyboard and paper tape records are variable length and are terminated by a carriage return and line feed in that order. The character ← may be used to clear the input buffer and reset to column 1. Illegal characters are ignored. For keyboard input the teletype bell is rung to notify the operator that bcd input is required.

Card records are a fixed length of 80 characters. Illegal characters are treated as question marks and cause a DATA error unless contained within a Hollerith field.

Magnetic tape records are a fixed length of 84 characters, and should be card or paper tape images with blank padding characters. The special character ← is permitted as defined for paper tape. Illegal characters are treated as blanks.

It should be noted that the model-33 teletype paper tape punch must be turned on and off by the operator.

SECTION IX

GLOSSARY

actual argument - an argument contained in a function reference or CALL statement

alphanumeric character - an alphabetic or numeric character

argument - a parameter used to pass data between programs and procedures

arithmetic expression - a sequence of constant, variable, or function references connected by arithmetic operators

arithmetic operator - one of the following characters with its associated connotation:

+ (addition)
- (subtraction)
* (multiplication)
/ (division)
** (exponentiation)

array - an ordered set of data of one or two dimensions

array element - one of the members of the set of data of an array

array name - a name that is defined in a DIMENSION statement

column - a character position in a line

comment line - a line with the character C in column 1

continuation line - a line that contains any character other than the digit zero or the character blank in column 6 and that contains blank characters in columns 1 through 5. A continuation line may only follow an initial line or another continuation line.

constant - a name that references a value. A constant may not be redefined

data type - the type of data, either integer or real dummy - a dummy argument

dummy argument - an argument used to indicate data type, number, and order of procedure arguments.

end line - a program unit terminator

executable program - a main program with possible one or more subprograms

executable statement - a statement that specifies an action of the program. An arithmetic assignment statement, control statement, or input-output statement

expression - an arithmetic expression

external procedure - a subprogram external to a program unit

FORTRAN character set - all alphanumeric and special characters listed on pages 1-1 and 1-2

function - a function subprogram, intrinsic function, or statement function

function reference - a function name followed by an actual argument list contained in parentheses

function subprogram - a FUNCTION statement followed by program body

initial line - a line that is neither a comment line nor an end line and that contains the digit zero or the character blank in column 6

integer - a datum which assumes only integral values. It may assume positive, negative, and zero values.

integer constant - a constant that references an integer value

integer variable - an integer datum that is identified by a symbolic name beginning with any one of the characters I, J, K, L, M, or N

line - a string of 72 characters each of which is a valid FORTRAN character. The character positions in a line are called columns and are consecutively numbered from left to right beginning with column 1.

list - a set of identifiable elements, each of which is separated from its successor by a comma

main program - a program body

name - an element of a statement which is used to reference objects such as data or procedures

non-executable statement - a statement that describes the characteristic and arrangement of data, editing information, statement functions, and classification of program units

operator - an element of a statement which specifies an action upon named objects

procedure - a function or subroutine

processor - the program which processes FORTRAN programs

program - a collection of statements, comment lines, and end lines

program body - a collection of optional specification statements optionally followed by statement function definition, followed by a program part, followed by an end line

program unit - a main program or subprogram

program part - at least one executable statement. A program part may but need not contain FORMAT statements

real - a datum which is a processor approximation to the value of a real number. A real datum assumes both integral and fractional values and may assume positive, negative, and zero values

real constant - a constant that references a real value

real variable - a datum that is identified by a symbolic name beginning with any character other than I, J, K, L, M, or N

reference - a verb indicating an identification of a datum and implying that the current value of the datum will be made available signed constant - a constant preceded by a plus or minus sign

special character - one of the ten characters: blank, equals, plus, minus, asterisk, slash, left parenthesis, right parenthesis, comma, and decimal point.

specification statement - a COMMON, DIMENSION, or EQUIVALENCE statement

statement - an initial line optionally followed by up to five ordered continuation lines. The statement is contained in columns 7 through 72 of the lines

statement label - one to four digits, the value of which must be greater than zero. Leading zeros are not significant

string - a series of data

subprogram - a SUBROUTINE or FUNCTION statement followed by a program body containing at least one RETURN statement

subroutine - a subroutine subprogram

subroutine subprogram - a SUBROUTINE statement followed by a program body

subscript - a parenthesized list of subscript expressions

subscript expressions - any one of the following expressions: C*V+K, C*V-K, C*V, V+K, V-K, V, K, where C and K are integer constants and V is an integer variable reference

symbolic name - one to five alphanumeric characters, the first of which must be alphabetic

variable - a datum that is identified by a symbolic name

SUBROUTINE DESCRIPTIONS

SECTION I

GENERAL DESCRIPTION

1.1 INTRODUCTION

This manual is one in the series of functional publications for the DATA 620/i computers. It is intended to acquaint the programmer with the standard subroutine library and how it is used. The manual is divided into the following four areas:

- Programmed Arithmetic
- Elementary Functions
- Utility and Debugging Routines
- Executive Routines

Each routine is documented in accordance with the programming standards as set forth in the following pages. These standards show the various categories and how they are documented.

It will be most helpful for each programmer to read over the standards before using any of the standards library; in addition, it will be helpful if the standards are followed when writing programs and submitting programs to the users group.

1.2 PROGRAMMING STANDARDS

1.2.1 Memory Allocations

Computer locations X7756 through X7777 octal will be used for various bootstraps, e.g., short programs for loading in the first record of a service library or service library loader from paper tapes or discs, where x = 0 for 4096 words and x = 1 for 8192 words. Routines will be distributed in relocatable binary or symbolic assembly language.

1.2.2 Subroutine Entry and Exit

If a subroutine requires only one parameter or argument, programmed entry will be made by first loading the desired parameter into the A register and then executing a return-jump to the subroutine.

Where more than two input parameters are required, the parameter will be entered into the program following the return-jump to the subroutine. The following sequence of instructions will be used:

LOCATION	INSTRUCTION	REMARKS
P	Return jump	Return jump to subroutine
P + 2	Parameter	Parameters or parameter locations for subroutine

LOCATION	INSTRUCTION	REMARKS
P + 3	Parameter	Parameters or parameter locations for subroutine
P + 4	Parameter	Parameters or parameter locations for subroutine
P + n	Parameter	Parameters or parameter locations for subroutine
P + n + 1	Jump to error	To execute error action
P + n + 2	Normal return	Continuation of program.

SECTION II

PROGRAM DESCRIPTION

2.1 INTRODUCTION

The published material for each routine will constitute a distinct package, separated materially from all other routines. (This is done to facilitate revisions and re-publication of the material for one routine without the necessity of re-publishing all others.) The published material for each routine will be as follows:

a. Identification

Title
Identification
Category
Programmer
Date

b. Purpose

c. Use

Calling sequence or operational procedure
Arguments or parameters
Space required (decimal)
Temporary storage requirements (decimal)
Alarms or printouts
Error returns or error codes
Error stops
Input and output devices
Input and output formats
Sense switch settings
Timing
Accuracy
Cautions to users
Equipment configuration
References

d. Method of Algorithm

e. Flow Charts

If any of the previous items are not applicable in the routine, the words "not applicable" will be inserted.

2.2 IDENTIFICATION

Each program will be identified by a category designator consisting of the following parts: classification code, program identification, and title.

The classification code will consist of a letter, indicating the primary class, followed by a digit indicating the subclass, chosen from the following expandable list:

a. Programmed arithmetic. Real (fixed point, double precision).

b. Elementary functions.

    Trigonometric
    Exponential and logarithmic
    Hyperbolic
    Roots and powers

c. Input

    Binary
    Octal
    Alphanumeric

d. Output

    Binary
    Octal
    Alphanumeric

e. Executive Routines.

    Assembly
    Compiling

f. Debugging Routines.

    Tracing
    Dump
    Search
    Breakpoint

g. Diagnostic programs.

h. Service programs.

    Clear
    Check sum programs
    Restore, rewind, bootstrap programs

i. All others.

SECTION III
PROGRAMMED ARITHMETIC

This section contains programmed routines separated into distinct packages. Each routine will follow the format described in section II, program description. As new routines are developed, they can easily be inserted into the proper section.

IDENTIFICATION

Title: Fixed single-precision integer binary-to-decimal conversion

Identification: XBTD

Category: A1

Programmer: J.H. Hathwell

Date: October, 1965

PURPOSE

XBTD converts the absolute value of the integer in the A register, modulo 10,000, to a four digit decimal coded integer in the B register. The input is retained in the A register and the X register is unchanged. The output range is 0 through 9999 inclusive

USE

1. Calling Sequence

   Call XBTD

2. Arguments or Parameters

   The binary argument is in the A register before and after execution.

3. Space Required

   Twenty-seven words.

4. Temporary Storage Requirements

   Four words.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   None.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Maximum: 138 cycles.
    Average: 137 cycles.
    Minimum: 136 cycles.

12. Accuracy

    Exact.

13. Cautions to User

    -2^{15} causes overflow and a meaningless result.

14. Equipment Configuration

    Not applicable.

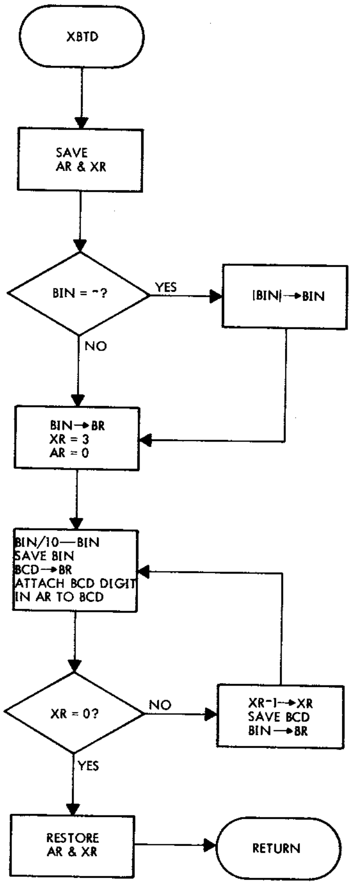
15. References

    Not applicable.

METHOD

Successive division of binary integer by 10_{10} with concatenation or remainders.

FLOW CHART



IDENTIFICATION

Title: Fixed single-precision integer decimal-to-binary conversion

Identification: XDTB

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDTB converts the four-digit decimal-coded integer in the A register to a binary integer in the B register. The input is retained in the A register with the X register unchanged. The output range is +0 through +9999 inclusive.

USE

1. Calling Sequence

   Call XDTB

2. Arguments or Parameters

   The decimal argument is in the A register before and after execution.

3. Space Required

   Twenty-four words.

4. Temporary Storage Requirements

   Four words.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   None.

7. Error Stops

None.

8. Input and Output Devices

Not applicable.

9. Input and Output Format

Not applicable.

10. Sense Switch Settings

Not applicable.

11. Timing

113 cycles.

12. Accuracy

Exact.

13. Cautions to Users

Input is not checked for legal bcd codes, but is evaluated as:

$$D_{3*} 10^3 + D_{2*} 10^2 + D_{1*} 10^1 + D_{0*} 10^0$$

where D is a four-bit binary number.

14. Equipment Configuration

Minimum configuration.

15. References

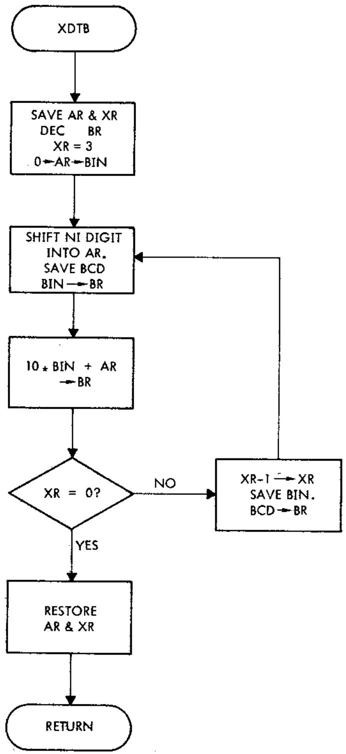
Not applicable.

METHOD

Successive multiplication of digits by powers of 10 with accumulation.

$$B = ((10D_3 + D_2)\ 10 + D_1)\ 10 + D_0\ .$$

FLOW CHART



IDENTIFICATION

Title: Fixed-point single-precision multiply

Identification: XMUL

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XMUL provides the software version of the (optional) hardware multiply instruction.

USAGE

1. Calling Sequence

   LDB Multiplier
   LDA Constant
   CALL XMUL
   PZE Address of multiplicand
   Normal return.

2. Arguments or Parameters

   On entry: A = constant to be added to product at 2^{30},
   B = multiplier.

   On exit: A, B = double-precision product,
   X is unchanged.

3. Space Required

   38 words (046_8).

4. Temporary Storage Required

   2 words.

5. Alarms or Printouts

None.

6. Error Returns or Codes

ØV is set (1) if the product is greater than 2**(NBIT-1)-1.

7. Error Stops

None.

8. Input and Output Devices

None.

9. Input and Output Formats or Tables

None.

10. Sense Switch Settings

None.

11. Timing

Maximum:	(B = 1)	436.75 cycles
Average:		404.75 cycles
Minimum:		372.75 cycles

12. Accuracy

Exact.

13. Cautions to User

None.

14. Equipment Configuration

Minimum.

15. References

DATA 620/i system reference manual (MUL).

METHOD

Recursive addition of multiplicand with shifting.

IDENTIFICATION

Title: Fixed-point single-precision divide

Identification: XDIV

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDIV provides the software version of one (optional) hardware divide instruction. The true remainder and quotient are delivered to the A register and B register, respectively.

USE

1. Calling Sequence

   LDA (high dividend)
   LDB (low dividend)
   Call XDIV
   PZE (address of divisor)
   Normal return.

2. Arguments or Parameters

   On entry: A, B = double precision dividend.

   On exit: A = remainder, B = quotient, X is unchanged.

3. Space Required

   70 words (0106_8).

4. Temporary Storage Required

   5 words.

5. Alarms or Printouts

   None.

6. Error Returns or Codes

   ØV is set (1) if the dividend is not less than the divisor.

7. Error Stops

   None.

8. Input and Output Devices

   None.

9. Input and Output Formats or Tables

   None.

10. Sense Switch Settings

    None.

11. Timing

    Average: 200 cycles

12. Accuracy

    Exact.

13. Cautions to User

    This routine produces the true quotient and remainder, i.e., -2/1 = quotient of -2 and remainder of zero.

14. Equipment Configuration

    Minimum.

15. References

    DATA 620/i system reference manual (DIV).

METHOD

Unsigned, non-restoring divide algorithm.

IDENTIFICATION

Title: Fixed-point double-precision 2's complement

Identification: XDCO

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDCO takes the 2's complement of the double-precision number in the A register and B register. The X register is unchanged.

USE

1. Calling Sequence

   Call XDCO

2. Arguments or Parameters

   The A register and the B register contain the double-precision argument before and the 2's complement after execution.

3. Space Required

   Thirteen words.

4. Temporary Storage Requirements

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   None.

7. Error Stops

None.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats

Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

10. Sense Switch Settings

Not applicable.

11. Timing

9.5 cycles.

12. Accuracy

Exact.

13. Cautions to Users

XDCO may set the overflow register.

14. Equipment Configuration

Not applicable.

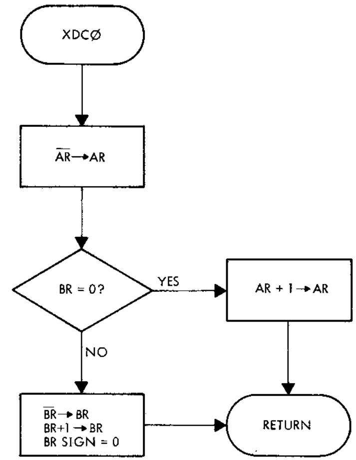
15. References

Not applicable.

METHOD

The argument is complemented and the low-order bits are tested for a carry condition.

FLOW CHART



IDENTIFICATION

Title: Fixed-point single-precision add

Identification: XDAD

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDAD adds a double-precision number whose high-order address is in the calling sequence to the double-precision numbers in the A register and B register. The X register is unchanged.

USE

1. Calling Sequence

   Call XDAD
   PZE is the address of high-order bits of the double-precision augend.
   Normal return.

2. Arguments or Parameters

   The A register and the B register contain the double-precision addend before, and the double-precision sum after execution.

3. Space Required

   Twenty-one words.

4. Temporary Storage Requirements

   Two words.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   The overflow is set if a double-precision overflow occurs.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

10. Sense Switch Settings

    Not applicable.

11. Timing

    30 cycles.

12. Accuracy

    Exact.

13. Cautions to Users

    The sign of the low-order words of each double precision argument must be zero to generate the proper carry. Overflow flip-flop is set on an overflow.

14. Equipment Configuration

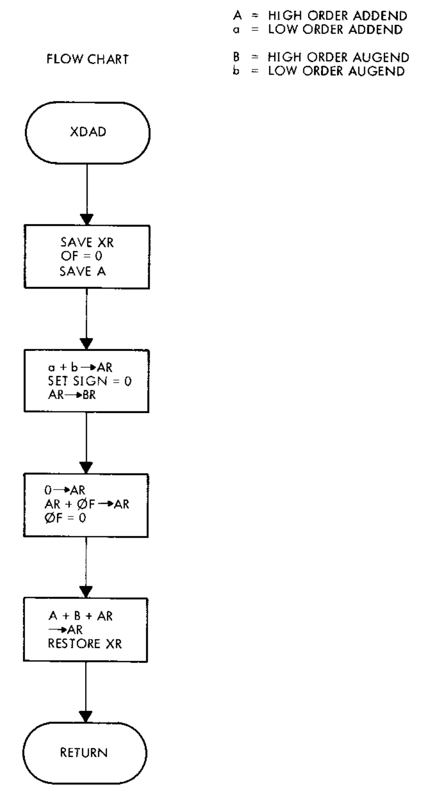
    Minimum configuration.

15. References

    Not applicable.

METHOD

Low-order words are added first and any carry generated is added to the high-order sum.



IDENTIFICATION

Title: Fixed-point double-precision subtract

Identification: XDSU

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDSU subtracts a double-precision number whose high-order address is in the calling sequence from the double-precision number in the A register and the B register. The X register is unchanged.

USE

1. Calling Sequence

   Call XDSU
   PZE is the address of high-order bits of the double-precision minuend.
   Normal return.

2. Arguments or Parameters

   The A register and the B register contain the double-precision subtrahend before and the double-precision difference after execution.

3. Space Required

   Twenty-three words.

4. Temporary Storage Requirements

   Two words.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   The overflow is set if a double-precision overflow occurs.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

10. Sense Switch Settings

    Not applicable.

11. Timing

    32 cycles.

12. Accuracy

    Exact.

13. Cautions to Users

    The sign of the low-order words of each double-precision argument must be zero to generate the proper carry. Overflow flip-flop is set on an overflow.

14. Equipment Configuration

    Minimum configuration.

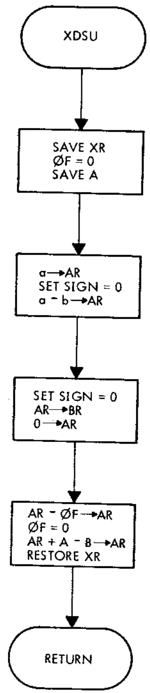
15. References

    Not applicable.

FLOW CHART

A = HIGH ORDER SUBTRAHEND
a = LOW ORDER SUBTRAHEND

B = HIGH ORDER MINUEND
b = LOW ORDER MINUEND



IDENTIFICATION

Title: Fixed-point double-precision multiply

Identification: XDMU

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDMU multiplies the double-precision number whose high-order address is in the calling sequence times the double-precision number in the A register and the B register. The X register is unchanged.

USE

1. Calling Sequence

   Call XDMU
   PZE is the address of the high-order bits of the multiplier.
   Normal return.

2. Arguments or Parameters

   The A register and the B register contain the double-precision multiplicand before and the double-precision product after execution.

3. Space Required

   Thirty-five words.

4. Temporary Storage Requirements

   Three words.

5. Alarms or Printouts

   None.

6. <u>Error Returns or Error Codes</u>

None.

7. <u>Error Stops</u>

None.

8. <u>Input and Output Devices</u>

Not applicable.

9. <u>Input and Output Formats</u>

Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

10. <u>Sense Switch Settings</u>

None.

11. <u>Timing</u>

71 cycles.

12. <u>Accuracy</u>

2^{-30} taken as a fraction.

13. <u>Cautions to Users</u>

Operands should be normalized to retain precision. Overflow is reset by XDMU.

14. <u>Equipment Configuration</u>

Uses hardware multiply.

15. <u>References</u>

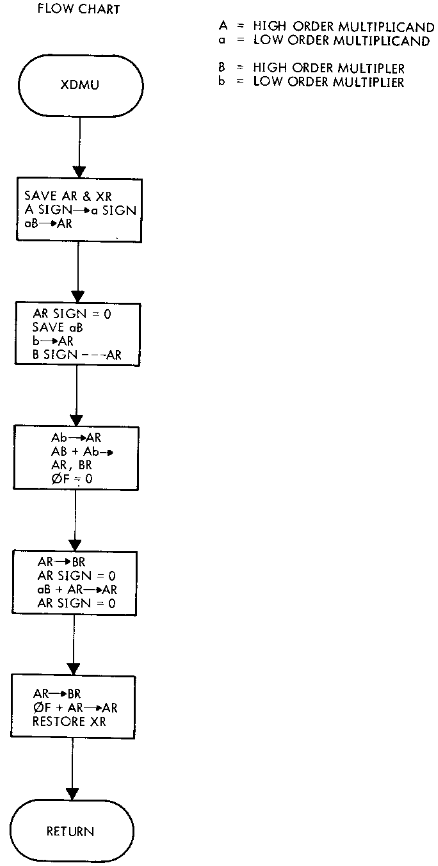
Not applicable.

METHOD

Double-precision addition of partial products.

$$(A + a)\ (B + b) \qquad AB{*}2^{0} + Ab{*}2^{-15} + aB{*}2^{-15}.$$

FLOW CHART



IDENTIFICATION

Title: Fixed-point double-precision divide

Identification: XDDI

Category: A1

Programmer: J. H. Hathwell

Date: October, 1965

PURPOSE

XDDI divides the double-precision number in the A register and B register by the double-precision number whose high-order address is in the calling sequence. The X register is unchanged.

USE

1. Calling Sequence

   Call XDDI
   PZE is the address of high-order bits of division.
   Normal return.

2. Arguments or Parameters

   The A register and B register contain the double-precision dividend before and the double-precision quotient after execution.

3. Space Required

   Fifty words.

4. Temporary Storage Requirements

   Five words.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

Overflow = 1 if a divide fault occurs.

7. Error Stops

None.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats

Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

10. Sense Switch Settings

Not applicable.

11. Timing

Both areas positive: 143 cycles.
Any areas negative: 172 cycles.

12. Accuracy

Accuracy is $\pm 2^{-29}$ taken as a fraction.

13. Cautions to User

Overflow is reset by XDDI. The dividend must be less than the divisor.

14. Equipment Configuration

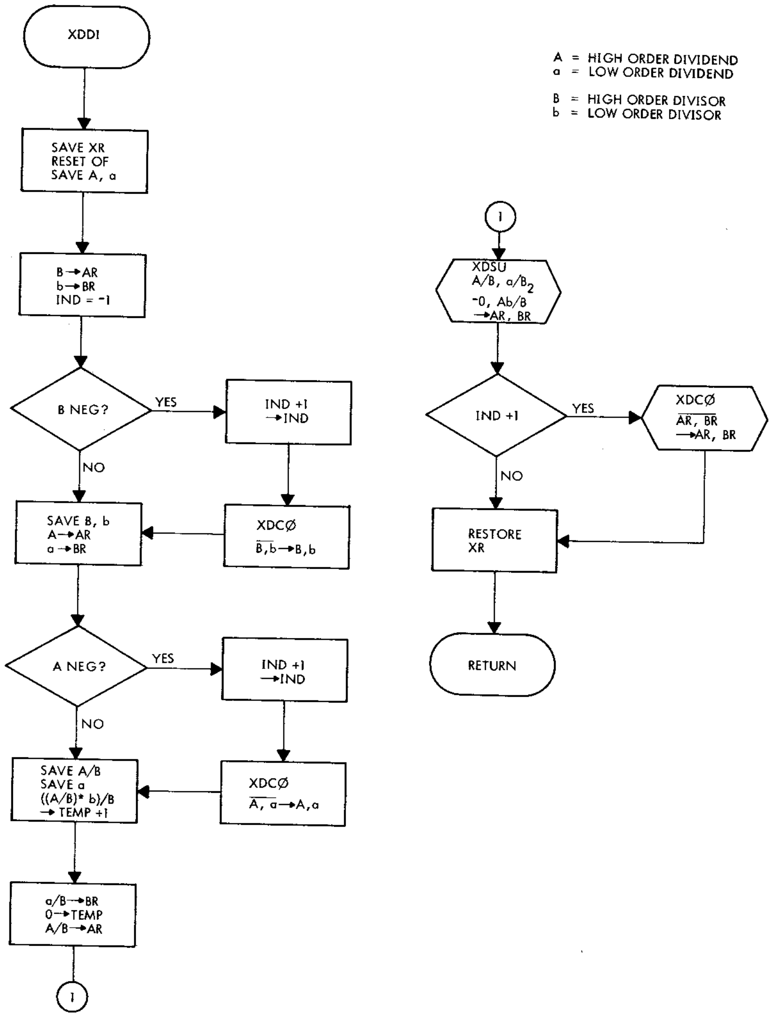
Hardware divide and multiply is used.

15. References

XDDI uses XDSU and XDCO.

METHOD

$$\frac{A + 9}{B + b} \approx \frac{A + 9}{B} - \frac{Ab}{B^2}$$



IDENTIFICATION

Title: Absolute value, floating point (type real)

Identification: ABS

Control Number: A56.00-1B.08.620

Programmer: M. McMillan

Date: November 4, 1965

PURPOSE

This routine takes the absolute value of the floating-point (real) quantity in the A, B registers, returning the result to the A, B registers. The absolute value of a is defined as -a if a was negative, as a if a was not negative.

USAGE

1. Calling Sequence

   Call ABS.

2. Arguments or Parameters

   Argument is in the A, B registers.

3. Space Required

   6 words.

4. Temporary Storage Required

   Not applicable.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

Not applicable.

10. Sense Switch Settings

Not applicable.

11. Timing

Minimum: 6 cycles.
Maximum: 9 cycles.

12. Accuracy

No loss of information.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

15. References

Not applicable.

METHOD

The method is explained by the coding itself:

LABEL	OPCODE	VARIABLE	COMMENTS
ABS	ENTRY		
	JAP*	ABS	Return immediately if not negative.
	CPA		One's complement high order word
	JMP*	ABS	if negative and return.

IDENTIFICATION

Title: Absolute value, fixed point (type integer)

Identification: IABS

Control Number: A58.00-1B.08.620

Programmer: M. McMillan

Date: November 4, 1965

PURPOSE

This routine takes the absolute value of the 16-bit signed integer in the A register and returns the result to the A register. The absolute value of a is defined as -a if the a was negative and a if a was non-negative.

USAGE

1. Calling Sequence

   Call IABS.

2. Arguments or Parameters

   The quantity in the A register is the argument. There are no other parameters.

3. Space Required

   7 words.

4. Temporary Storage Required

   Not applicable.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Maximum: 10 cycles.
    Minimum: 7 cycles.

12. Accuracy

    No loss of information.

13. Cautions to Users

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

    Not applicable.

METHOD

The method is explained by the subroutine code itself:

LABEL	OPCODE	VARIABLE	COMMENTS
IABS	ENTRY		
	JAP*	IABS	Return if argument positive or zero.
	CPA		If argument negative, one's complement
	IAR		and correct to two's complement.
	JMP*	IABS	Return.

IDENTIFICATION

Title: Transfer of sign, fixed point (type integer)

Identification: ISIGN

Control Number: A59.00-1B.08.620

Programmer: M. McMillan

Date: November 4, 1965.

PURPOSE

This routine applies the sign of the called (second) parameter to the quantity in the accumulator (first parameter). The parameters and result are fixed point quantities.

USAGE

1. Calling Sequence

   Call ISIGN, REF.

2. Arguments or Parameters

   The first parameter is located in the A register. The second parameter is located in core, whose address is in REF.

3. Space Required

   27 words, including two local working cells.

4. Temporary Storage Required

   Not applicable.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

Not applicable.

10. Sense Switch Settings

Not applicable.

11. Timing

Maximum: 39.75 cycles.
Minimum: 29.75 cycles.

12. Accuracy

No loss of information.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

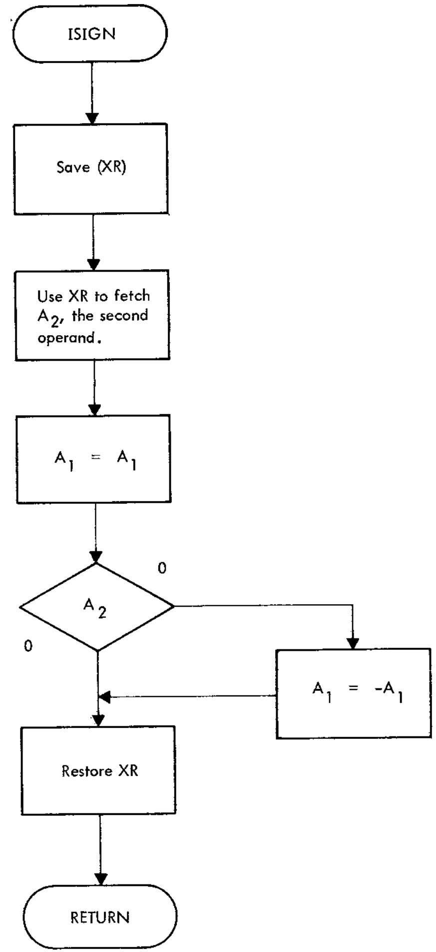
15. References

Not applicable.

METHOD

The method is illustrated by the flowchart. Uses $SE.

METHOD FLOW CHART



IDENTIFICATION

Title: Copy sign

Identification: SIGN

Control Number: A57.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To set sign of floating point number equal to that of argument.

USAGE

1. Calling Sequence

   Call SIGN, REF

2. Arguments or Parameters

   Floating point number in A, B registers. REF - address of argument.

3. Space Required

   17 words.

4. Temporary Storage Required

   2 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Floating point format.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 67.5 cycles.

12. Accuracy

    Exact.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

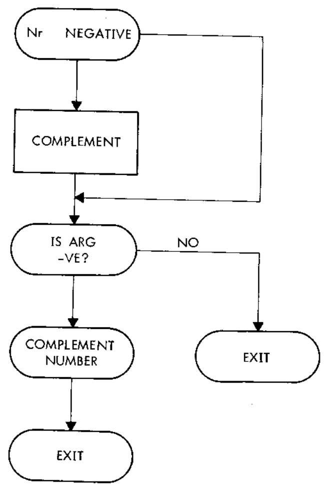
15. References

    Not applicable.

METHOD

Sets sign equal to that of argument. Output in A, B registers. Uses $SE.

FLOW CHART



IDENTIFICATION

Title: Separate mantissa

Identification: $FMS, $FSM

Control Number: 102.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To separate a positive floating point number into characteristic and mantissa.

USAGE

1. Calling Sequence

   Call $FMS or $FSM.

2. Arguments or Parameters

   A and B registers contain floating point number.

3. Space Required

   9 words.

4. Temporary Storage Required

   1 word.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

Floating point - input
A, B contain fixed point mantissa.
X contains characteristic at B8 on exit.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 13 cycles.

12. Accuracy

Exact.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

15. References

Not applicable.

METHOD

See listing.

Output in A, B (mantissa) and X (characteristic) registers.

IDENTIFICATION

Title: Floating point number to integer

Identification: $HS

Control Number: 103.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To convert a floating point number to an integer.

USAGE

1. Calling Sequence

Call $HS, STORE.

2. Arguments or Parameters

Number in A, B registers. STORE - address of memory where the result is to be saved.

3. Space Required

55 words.

4. Temporary Storage Required

1 word.

5. Alarms or Printouts

Not applicable.

6. Error Returns or Codes

If number greater than 2 * * 15 or less than 1, it exits with A, B registers set to zero.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

Floating point input. Fixed point integer output.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 89.5 cycles.

12. Accuracy

15 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

15. References

FORTRAN reference manual.

METHOD

Uses $SE. See listing.

IDENTIFICATION

Title: Normalize

Identification: $NML

Control Number: A54.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To normalize a double precision number.

USAGE

1. Calling Sequence

Call $NML.

2. Arguments or Parameters

Number in A, B registers.

3. Space Required

39 words.

4. Temporary Storage Required

2 words.

5. Alarms or Printouts

Not applicable.

6. Error Returns or Codes

Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

Fixed point format.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 101 cycles.

12. Accuracy

22 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

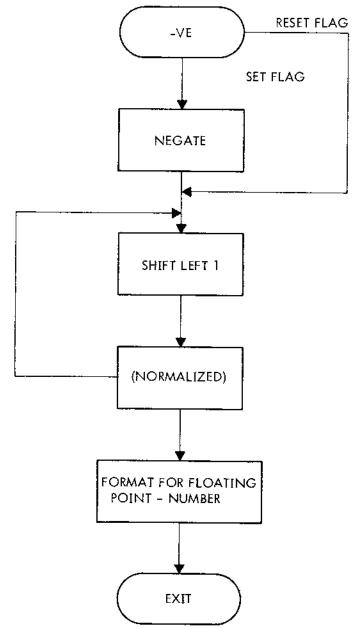
15. References

FORTRAN reference manual.

METHOD

Shifts to sign and tests for sign set. Uses XDCO. Output in A, B registers. Flag for sign in X register.

FLOW CHART



IDENTIFICATION

Title: Floating add

Identification: $QK

Control Number: A51.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To add 2 floating point numbers.

USAGE

1. Calling Sequence

   Call $QK, REF.

2. Arguments or Parameters

   A, B registers contain first argument. Ref - address of second argument. Result in A, B registers.

3. Space Required

   140 words.

4. Temporary Storage Required

   9 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   See floating point format.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 224 cycles.

12. Accuracy

    22 bits.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

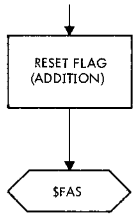
    FORTRAN reference manual.

METHOD

Algebraically adds two numbers.

$QK and $QL use common logic $FAS. $FAS determines if it is a arithmetic addition or subtraction and proceeds accordingly. $FAS has a special entry linkage and is used solely by $QK and $QL.

FLOW CHART



IDENTIFICATION

Title: Floating subtract

Identification: $QL

Control Number: A52.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To compute difference of two floating point numbers.

USAGE

1. Calling Sequence

   Call $QL, REF.

2. Arguments or Parameters

   Minuend in A, B registers. REF - address of first word of subtrahend.

3. Space Required

   4 words.

4. Temporary Storage Required

   Not applicable.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

See floating point format.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 223 cycles.

12. Accuracy

22 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

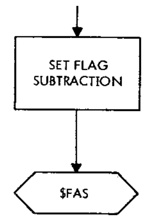
15. References

FORTRAN reference manual.

METHOD

Uses $QK.

FLOW CHART



IDENTIFICATION

Title: Floating add or subtract

Identification: $FAS

Control Number: A53.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To provide common logic for $QK, $QL. It has a special linkage for use by $QK or $QL.

USAGE

1. Calling Sequence

   Not for general use.

2. Arguments or Parameters

   Not applicable.

3. Space Required

   Included in $QK.

4. Temporary Storage Required

   Not applicable.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: included with $QK and $QL.

12. Accuracy

    Exact.

13. Cautions to User

    Not for general use.

14. Equipment Configuration

    Not applicable.

15. References

    $QK, $QL

METHOD

See listing.

IDENTIFICATION

Title: Floating-point multiply or divide

Identification: $QM, $QN

Control Number: A55.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To multiply 2 floating point numbers. To divide one number by another.

USAGE

1. Calling Sequence

   Call $QM, REF for multiply.
   Call $QN, REF for divide.

2. Arguments or Parameters

   REF - address of multiplier or divisor.

3. Space Required

   126 words.

4. Temporary Storage Required

   7 words.

5. Alams or Printouts

   Not applicable.

6. Error Returns or Codes

   If divisor = 0, A, B registers set to zero and overflow on.

   If result is less than 2 * * (-200_8) or greater than 2 * * ($+177_8$), it returns with 0 in A, B registers and overflow on.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Floating point format. Output in A, B registers.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 237, multiply.
    334, divide.

12. Accuracy

    22 bits multiply.
    21 bits divide.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

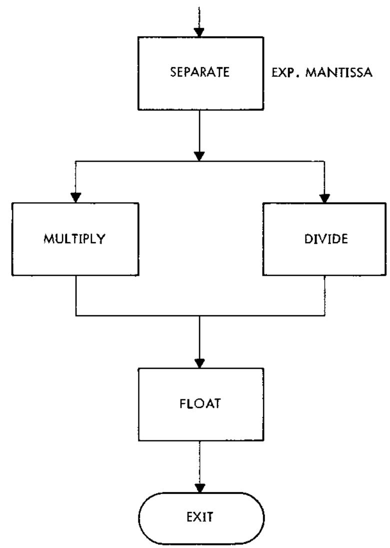
15. References

    FORTRAN reference manual.

METHOD

Separate the mantissa and use XDMU for multiply or XDDI for divide. Uses $FMS, $SE.

FLOW CHART



IDENTIFICATION

Title: Integer number to floating-point number

Identification: $QS

Control Number: 101.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To float an integer.

USAGE

1. Calling Sequence

   Call $QS, STORE.

2. Arguments or Parameters

   Argument in A register. STORE - address of memory where result is to be saved.

3. Space Required

   36 words.

4. Temporary Storage Required

   3 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Floating-point format output. Fixed-point integer input.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 138 cycles

12. Accuracy

    Exact.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

    FORTRAN reference manual

METHOD

Formats the absolute number to floating point and adjusts sign according to input. Uses $SE.

SECTION IV

ELEMENTARY FUNCTIONS

This section contains programmed routines separated into distinct packages. Each routine will follow the format described in section II, program description. As new routines are developed, they can be easily inserted into the proper section.

IDENTIFICATION

Title: Fixed single-precision logarithm

Identification: XLOG

Category: B2

Programmer: M.C. McMillan

Date: June, 1965

PURPOSE

XLOG computes the natural logarithm of 1 + x, where the single-precision quantity x is in the A register. If

$$0 < X < 1,$$

the result is returned to the A register, otherwise an error exit is taken without further action. Input and output are scaled by 2^0.

USE

1. Calling Sequence

   JMPM XLOG
   JMP (error procedure)
   Normal return.

2. Arguments or Parameters

   The argument x is placed in A before calling XLOG.

3. Space Required

   Eighteen words.

4. Temporary Storage Requirements

   None

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   Error return if x is negative.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Maximum: 203 cycles
    Average: 203 cycles
    Minimum: 9 cycles

12. Accuracy

    Error is less than 2^{-14} machine scale.

13. Cautions to Users

    Routine XLOG calls subroutine POLY.

14. Equipment Configuration

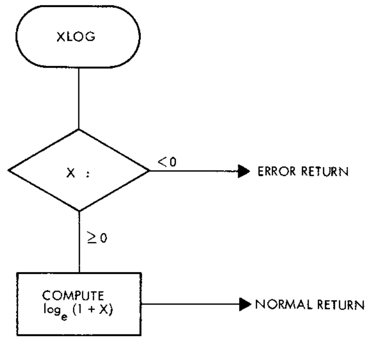
    Not applicable.

15. References

    Not applicable.

METHOD

XLOG uses a Chebychev polynomial of the fifth degree.



IDENTIFICATION

Title: Fixed single exponential, positive argument

Identification: XEXP

Category: B2

Programmer: M.C. McMillan

Date: June, 1965

PURPOSE:

XEXP computes the exponential of X, located in the A register:

$$e^x, \ 0 \leq x < 1$$

e^x is scaled 2^{-2}. The result is placed in the A register. (Also see PURPOSE in subroutine XEXN.)

USE

1. Calling Sequence

   JMPM XEXP
   JMP (error return)
   Normal return.

2. Arguments or Parameters

   The argument X is located in the A register prior to the call.

3. Space Required

   Seventeen words.

4. Temporary Storage Requirements

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   An error return is taken without the other action if the argument is negative.

7. Error Stops

None.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats

Not applicable.

10. Sense Switch Settings

Not applicable.

11. Timing

Normal: 187 cycles
Error return: 8 cycles

12. Accuracy

Error is less than 2^{-14} of machine scale.

13. Cautions to Users

Note relative scale between input and output, and that they differ from scales relative to the routine XEXN. System subroutine XEXN is called by XEXP.

14. Equipment Configuration

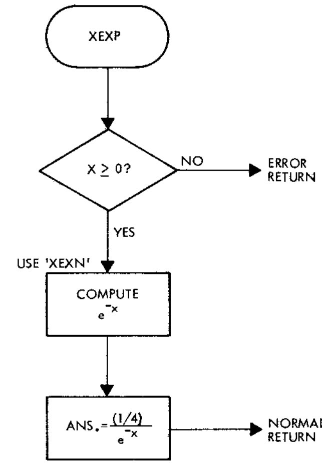
Not applicable.

15. Reference

Not applicable.

METHOD

The exponential is performed by means of a Chebychev polynomial of the fifth degree.



IDENTIFICATION

Title: Fixed single exponential, negative argument

Identification: XEXN

Category: B2

Programmer: M.C. McMillan

Date: June, 1965

PURPOSE

XEXN computes the exponential of x, located in the A register:

$$e^{x}, \; -1 < x \leq 0$$

e^{x} is scaled x 2^{0}. The result is placed in the A register. (Also see purpose in subroutine XEXP.) The exponential was split into two subroutines, XEXP and XEXN, to increase scaling flexibility.

USE

1. Calling Sequence

   JMPM XEXN
   JMP (error procedure)
   Normal return.

2. Arguments or Parameters

   The argument x is located in the A register prior to the call.

3. Space Required

   Eighteen words.

4. Temporary Storage Requirements

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   An error return is taken without other action if the argument is negative.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Normal (maximum): 159 cycles
    Error return: 8 cycles

12. Accuracy

    Error is less than 2^{-14} of machine scale.

13. Cautions to Users

    Note that scaling conventions differ between subroutines XEXN and XEXP.

14. Equipment Configuration

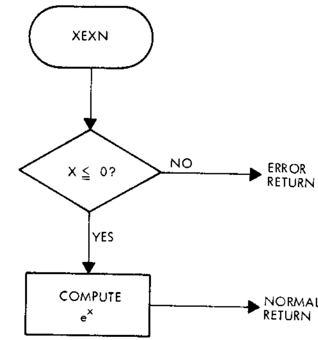
    Not applicable.

15 References

   Not applicable.

METHOD

The exponential is performed by means of a Chebychev polynomial of the fifth degree. ,e.



IDENTIFICATION

Title: Fixed single-precision square root (short)

Identification: XSQT

Category: B4

Programmer: M.C. McMillan

Date: October, 1965

PURPOSE

XSQT takes the unrounded square root of the quantity in the A register if it is non-negative. The result is returned to the A register. The A register is unchanged if the input is negative. XSQN is recommended instead unless there is a hardware divide option.

USE

1. Calling Sequence

   JMPM XSQT
   JMP (error procedure)
   Normal return.

2. Arguments or Parameters

   The argument is located in the A register before execution.

3. Space Required

   Forty-three words (forty-four if no automatic divide).

4. Temporary Storage Requirements

   Five words.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   Error return if argument is negative.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    276 cycles (hardwire divide; otherwise add 15 times the software divide time for maximum cycle time).

12. Accuracy

    Error is less than 1.5×2^{-15} machine scale.

13. Cautions to Users

    None.

14. Equipment Configuration

    Not applicable.

15. References

    Not applicable.

METHOD

Uses Newton-Ralphson formula:

$$X_i + 1 = 1/2\, X_i + \frac{A}{2X_i},\ \lim X_i = \sqrt{A}$$

in the form

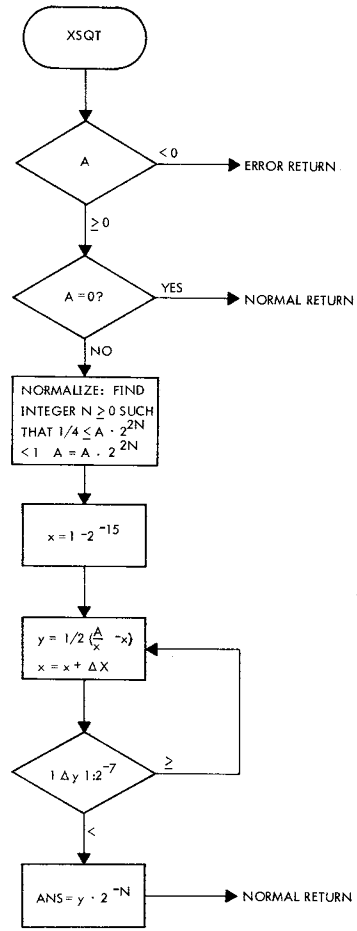
$$X_i + 1 = X_i + \Delta X_i$$

where

$$\Delta X_i = 1/2 \left(\frac{A}{X_i} = X_i \right)$$

IF $X_0 = 1 - 2^{-15}$ (the maximum positive numeric value of a number in a 16-bit binary representation) then $\Delta X_i \leq 0$ for all steps.

IF $|\Delta X_i| < 2^{-7} - 2^{-15}$ at a given step, there is no need to take another step, as would be required if testing differences of successive x-estimates. A maximum of four divide operations makes XSQT less attrative than XSQN (only one divide and one short-word multiply) unless automatic divide-hardware is present.



IDENTIFICATION

Title: Fixed single-precision sine

Identification: XSIN

Category: B1

Programmer: M.C. McMillan

Date: August, 1965

PURPOSE

XSIN takes the sine of the quantity X in the A register for range $-\pi \leq x \leq \pi$. The input is scaled by 2^{-2}. The output is returned to the A register.

USE

1. Calling Sequence

   Call XSIN

2. Arguments or Parameters

   The argument X is in the A register.

3. Space Required

   Thirty-one words.

4. Temporary Storage Requirements

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   None.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Maximum is typical: 175 cycles.

12. Accuracy

    Error is less than 2^{-14} machine scale.

13. Cautions to Users

    XSIN requires subroutine POLY. No test is made for $\pi \; |x| \; 4$.

14. Equipment Configuration

    Not applicable.

15. References

    Not applicable.

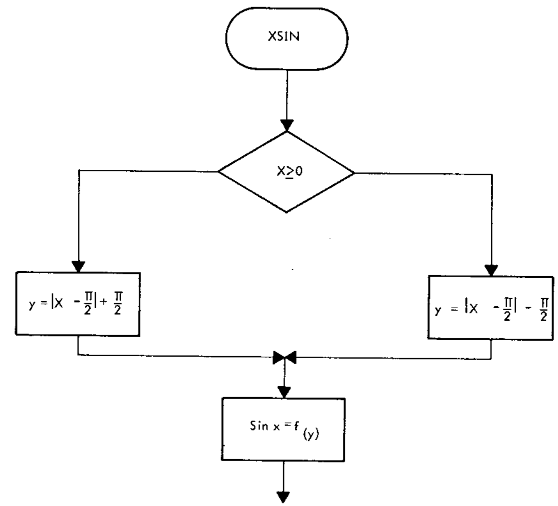
METHOD

Uses a change of variable to y to reduce range from $(-\pi, \pi)$ to $(-\pi/2, \pi/2)$. The change of variable is $\sin x = \sin y$.

$$y = |X - \frac{\pi}{2}| \; \frac{-\pi}{2} \qquad \text{if } X \geq 0$$

$$y = |X - \frac{\pi}{2}| \; \frac{-\pi}{2} \qquad \text{if } X < 0$$

The Taylor sine series, truncated to five items, is used for sin y.



IDENTIFICATION

Title: Fixed single-precision cosine

Identification: XCOS

Category: B1

Programmer: M.C. McMillan

Date: August, 1965

PURPOSE

XCOS takes the cosine of the quantity X in the A register from range $-\pi \leq X \leq \pi$. The input is scaled by 2^{-2} and the output is scaled by 2^{-1}. The output is returned to the A register.

USE

1. Calling Sequence

   Call XCOS

2. Arguments or Parameters

   The argument X is in the A register.

3. Space Required

   Twenty words.

4. Temporary Storage Requirements

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   None.

7. Error Stops

   None.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Maximum is typical: 172 cycles.

12. Accuracy

    Error is less than 2^{-14} machine scale.

13. Cautions to Users

    XCOS requires subroutine POLY, no test is made for $\pi \; |X| \; 4$.

14. Equipment Configuration

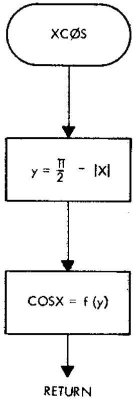
    Not applicable.

15. References

    Not applicable.

METHOD

Uses a change of variable to y in order to reduce the range of the variable from $(-\pi, +\pi)$ to $\frac{-\pi}{2}, \frac{+\pi}{2}$. Them $\cos x = \sin y$. Where $y = \frac{\pi}{2} - |X|$
The Taylor sine series, truncated to five terms is used for sin y.



IDENTIFICATION

Title: Fixed single-precision arctangent

Identification: XATN

Category: B1

Programmer: M.C. McMillan

Date: June, 1965

PURPOSE

XATN takes the arctangent of the quantity X in the A register, where $-1 < X < 1$. The input is scaled times 2^0 and the output is scaled times 2^0.

USE

1. Calling Sequence

   JMPM, XATN

2. Arguments or Parameters

   The argument X is in the A register.

3. Space Required

   Fifteen words.

4. Temporary Storage Requirements

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Error Codes

   None.

7. Error Stops

None.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats

Not applicable.

10. Sense Switch Settings

Not applicable.

11. Timing

Fixed: 211 cycles.

12. Accuracy

Error is less than 2^{-14} machine scale.

13. Cautions to Users

XATN requires system subroutine POLY.

14. Equipment Configuration

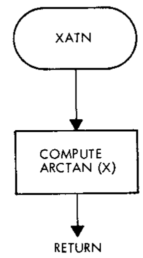
Not applicable.

15. References

Not applicable.

METHOD

XATN uses a Chebychev polynomial of seven terms. This polynomial is adequate for an 18-bit configuration.



IDENTIFICATION

Title: Single-precision polynomial

Identification: POLY

Category: B

Programmer: M.C. McMillan

Date: June, 1965

PURPOSE

POLY is a resident utility routine intended primarily to support the fixed-point single-precision mathematical subroutines requiring the evaluation of a polynomial in one variable of any finite degree.

USE

1. Calling Sequence

Call POLY (list of coefficients, format as below):

a. Type code
b. List of non-zero coefficients of degree greater than 1
c. Zero
d. Coefficient of degree 1
e. Coefficient of degree 0
f. Normal return

2. Arguments or Parameters

The type code is either 0 or 1. Zero denotes a polynomial in all powers; one denotes a polynomial in either odd or even powers.

The list of coefficients of degree greater than one is written highest power first, and may be of any number. d) and e) coefficients must be present. Use zero to represent an absent term.

3. Space Required

Forty-six words.

4. Temporary Storage Requirements

Three words.

5. Alarms or Printouts

None.

6. Error Returns or Error Codes

None.

7. Error Stops

None.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats

Not applicable.

10. Sense Switch Settings

Not applicable.

11. Timing

40 memory cycles.
+16 memory cycles if code = 1.
+11 memory cycles if coefficient of degree 1 is not 0.
+23 memory cycles per term of degree greater than 1.

12. Accuracy

The accuracy attainable is close to unrounded full single-word precision. However, accuracy obtained depends upon correct techniques of scaling and may depend on mathematical characteristics of the polynomial being evaluated.

13. Cautions to Users

No action is taken if an additive overflow occurs during computation of the polynomial. Certain arbitrary combinations of coefficients may sharply reduce the accuracy attained. Missing interior coefficients of degrees higher than 1 must be approximated by small non-zero numbers, unless their absence is implied by type code = 1.

14. Equipment Configuration

Not applicable.

15. References

Not applicable.

METHOD

The polynomial is evaluated in Horner form. For example:

$$C_4 x^4 + C_3 x^3 + C_2 x^2 + C_1 x + C_0$$

is evaluated as:

$$(((C_4 x + C_3) x + C_2) x + C_1) x + C_0$$

The parameter list taking the forms 0, C_4, C_3, C_2, 0, C_1, C_0. The polynomial

$$C_7 x^7 + C_5 x^5 + C_3 x^3 + C_1 x$$

is evaluated as:

$$(((C_7 x^2 + C_5) x^2 + C_3) x^2 + C_1) x + 0$$

the parameter list taking the form: 1, C_7, C_5, C_3, 0, C_1, 0.

IDENTIFICATION

Title: Natural log of floating-point number

Identification: ALOG

Control Number: B24.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To compuete natural log of a floating-point number.

USAGE

1. Calling Sequence

   Call ALOG, REF

2. Arguments or Parameters

   REF - Address of argument.

3. Space Required

   125 words.

4. Temporary Storage Required

   8 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Exits to $ER if argument = 0

7. Error Stops

Not Applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

Floating-point format. Output in A, B registers.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 907.5 cycles.

12. Accuracy

21 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

15. References

FORTRAN reference manual.

METHOD

$$\log A = \log_2 A * \log_e 2$$

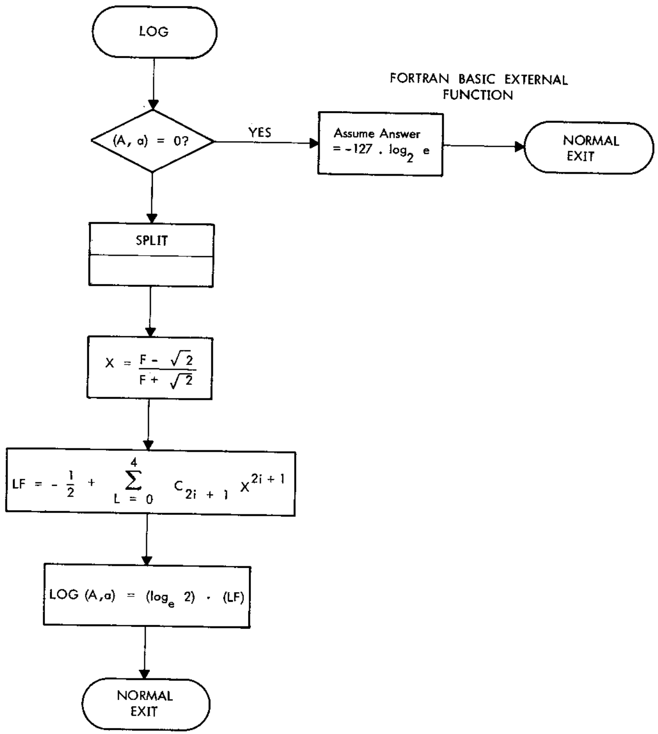
$$\log_2 A = -1/2 + \sum_{i=0}^{i=4} C_{2i+1} Z^{2i+1}$$

$$Z = \frac{F' - \sqrt{2}}{F' + \sqrt{2}}$$

$$A = F' * 2^b \text{ where } 1 \le F' < 2$$

C_{2i-1} are coefficients of series expansion. Uses $ER, $QS, $QK, $QM, XDMU, XDAD, $FMS, $NML, XDDI, XDSU, $SE routines.

FLOW CHART



IDENTIFICATION

Title: Arctangent of a floating-point number

Identification: ATAN

Control Number: B13.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

Computes arctangent of radians in floating point.

USAGE

1. Calling Sequence

   Call ATAN, REF.

2. Arguments or Parameters

   REF - address of the floating-point argument.

3. Space Required

   184 words.

4. Temporary Storage Required

   5 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats

Floating-point format.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 2888 cycles.

12. Accuracy

21 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

15. References

FORTRAN reference manual.

METHOD

Let $N = |X|$ or $N = |X/Y|$. The arctangent of N is evaluated by dividing the total range $0 < N < 10^{75}$ into three intervals; $(10^{-5}, \tan \pi/24)$, $(\tan \pi/24, 1)$, $(1, 10^8)$. If $N < 10^{-5}$, arctan N = N. If $N > 10^8$, arctan $N = \pi/2$.

The polynomial approximation in the interval $(10^{-5}, \tan \pi/24)$ is:

$$\mathrm{TAN}^{-1} N \approx C_1 N + C_2 N^3 + C_3 N^5$$

Continued fraction approximations are used in the remaining intervals.

Uses \$QM, \$QL, \$QN, \$QK, \$SE routines.

$$\mathrm{Tan}^{-1} N \approx N.\ A_1 + \cfrac{A_2}{(N^2 + B_2) - \cfrac{A_3}{(N^2 + B_3)}} \quad \text{in } (\tan \pi/24, 1)$$

and

$$\mathrm{Tan}^{-1} N \approx \pi/2 - N^{-1}\ D_1 - \cfrac{D_2}{(N^2 + E_2) - \cfrac{D_3}{(N^2 + E_3)}} \quad \text{in } (1, 10^8)$$

where

C_1 = 0 . 99999 99207
C_2 = 0 . 33329 66338
C_3 = 0 . 19574 08066
A_1 = 0 . 23882 29612
A_2 = 2 . 4452 05396
A_3 = 1 . 3247 47223
B_2 = 3 . 9435 29798
B_3 = 1 . 7982 49626
D_1 = 0 . 99999 92083
D_2 = 0 . 33328 70775
D_3 = 0 . 06355 00089
E_2 = 0 . 59859 98078
E_3 = 0 . 39535 44718

IDENTIFICATION

Title: Cosine

Identification: COS

Control Number: B12.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

Compute cosine of angle in floating-point radians.

USAGE

1. Calling Sequence

   Call COS, REF.

2. Arguments or Parameters

   REF - Address of first word of floating point number.

3. Space Required

   24 words.

4. Temporary Storage Required

   2 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Not applicable.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 1600 cycles.

12. Accuracy

    21 bits.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

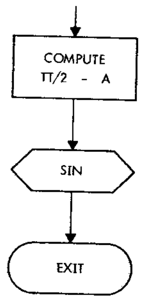
15. References

    FORTRAN reference manual.

METHOD

Computes sine of (π/2-A). Uses SIN, $QL, $SE. Output in A, B registers.

FLOW CHART



IDENTIFICATION

Title: Exponential

Identification: EXP

Control Number: B25.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To compute e**A. A - floating-point number.

USAGE

1. Calling Sequence

   Call EXP, REF

2. Arguments or Parameters

   REF - address of argument A.

3. Space Required

   230 words.

4. Temporary Storage Required

   2 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

See floating-point format.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 2750 cycles.

12. Accuracy

21 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

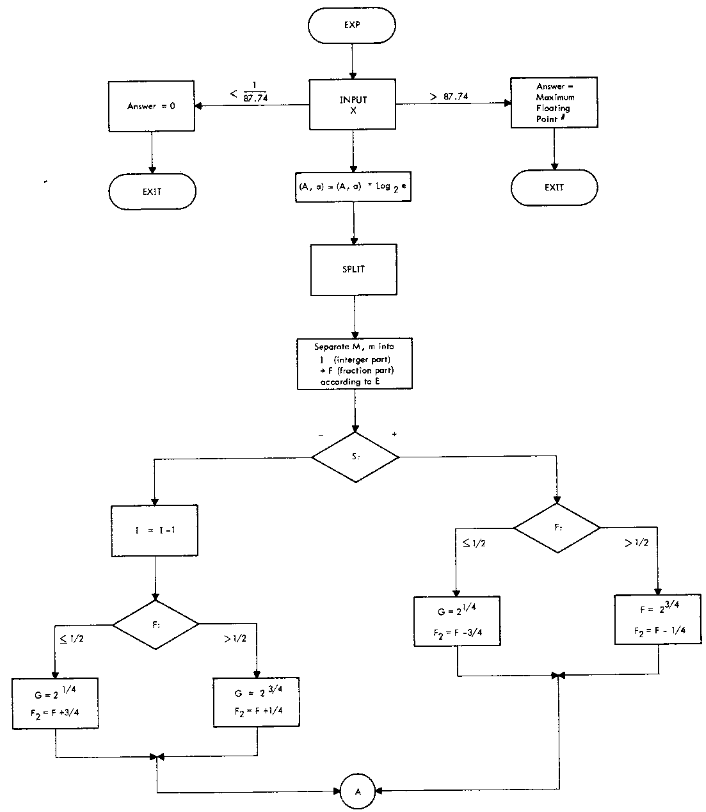
15. References

FORTRAN reference manual.

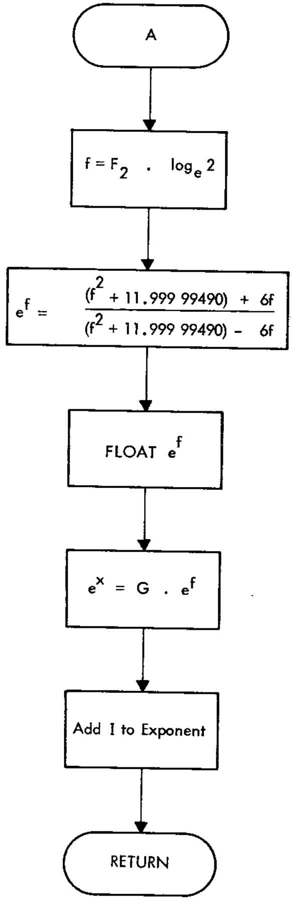
METHOD

Chebychev approximation uses XDMU, $QK, $QL, $QM, $QN, $SE.

FLOW CHART



FLOW CHART



IDENTIFICATION

Title: Sine

Identification SIN

Control Number: B11.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

Compute sine of radians in floating point.

USAGE

1. Calling Sequence

   Call SIN, REF

2. Arguments or Parameters

   REF - address (direct or indirect) of first word of a floating-point number.

3. Space Required

   151 words.

4. Temporary Storage Required

   6 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

Not applicable.

9. Input and Output Formats or Tables

See floating-point format.

10. Sense Switch Settings

Not applicable.

11. Timing

Average: 1305 cycles.

12. Accuracy

21 bits.

13. Cautions to User

Not applicable.

14. Equipment Configuration

Not applicable.

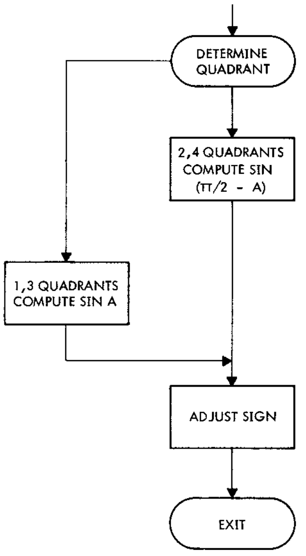
15. References

FORTRAN reference manual.

METHOD

First 5 terms of Taylor series expansion output in A, B registers. Uses $NML, $QM, XDMU, XDAD, $SE, $FMS.

FLOW CHART



IDENTIFICATION

Title: Square root

Identification: SQRT

Control Number: B41.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

Computer square root of a floating point number.

USAGE

1. Calling Sequence

   Call SQRT, REF.

2. Arguments or Parameters

   REF - address of the argument.

3. Space Required

   83 words.

4. Temporary Storage Required

   6 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Exits with zero in A, B of argument negative and sets overflow flip flop.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Floating-point format.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 333 cycles.

12. Accuracy

    21 bits.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

    FORTRAN reference manual.

METHOD

Newton iteration three times. Uses $SE, XDDI, $FMS.

IDENTIFICATION

Title: Exponentiation of two integers

Identification: $HE

Control Number: B22.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To compute I**J

USAGE

1. Calling Sequence

   Call $HE, REF.

2. Arguments or Parameters

   I in A register. REF - address of J.

3. Space Required

   20 words.

4. Temporary Storage Required

   2 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   Fixed-point integers.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 4500 cycles.

12. Accuracy

    15 bits.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

    FORTRAN reference manual.

METHOD

Floats I and uses $PE. Uses $SE, $QS, $HS, $PE.

IDENTIFICATION

Title: Exponentiation

Identification: $PE

Control Number: B21.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To compute A**I .

USAGE

1. Calling Sequence

   Call $PE, REF.

2. Arguments or Parameters

   Argument in A, B registers.
   REF - address of index I.

3. Space Required

   21 words.

4. Temporary Storage Required

   4 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9. Input and Output Formats or Tables

   See Floating Point.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 4200 cycles.

12. Accuracy

    20 bits.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

    FORTRAN reference manual.

METHOD

Uses $QS, $QE, and $SE. Floats I and goes to A**B ($QE).

IDENTIFICATION

Title: Exponentiation

Identification: $QE

Control Number: B23.00-1B.08.620

Programmer: M. C. Advani

Date: August 31, 1966

PURPOSE

To computer A**B.

USAGE

1. Calling Sequence

   Call $QE, REF.

2. Arguments or Parameters

   Argument A in A, B registers. REF - address of argument B.

3. Space Required

   34 words.

4. Temporary Storage Required

   3 words.

5. Alarms or Printouts

   Not applicable.

6. Error Returns or Codes

   Not applicable.

7. Error Stops

   Not applicable.

8. Input and Output Devices

   Not applicable.

9: Input and Output Formats or Tables

   Floating-point format.

10. Sense Switch Settings

    Not applicable.

11. Timing

    Average: 4000 cycles.

12. Accuracy

    20 bits.

13. Cautions to User

    Not applicable.

14. Equipment Configuration

    Not applicable.

15. References

    FORTRAN reference manual.

METHOD

A**B - antilog of B log A
- e**(B log A)

Uses ALOG, EXP, $SE.

SECTION V

UTILITY AND DEBUGGING ROUTINES

This section contains programmed routines separated into distinct packages. Each routine will follow the format described in section II, program description. As new routines are developed, they can be easily inserted into the proper section.

IDENTIFICATION

Title: AID II program

Identification: AID II

Category: E41.00 - 1B04.620

Programmer: John H. Hathwell

Date: August 30, 1966

PURPOSE

To provide on-line program debugging.

USE

1. Calling Sequence

   Run at X6000.

   Where X = 0 for 4K, X = 1 for 8K, etc.

2. Arguments or Parameters

   None.

3. Space Required

   637 octal.

4. Temporary Storage Required

   None

5. Alarms or Printouts

   Insert I.

6. Error Returns or Codes

   None

7. Error Stops

   None.

8. Input and Output Devices

   Model 33/35 A or B teletypes.

9. Input and Output Formats or Tables

   None.

9.A Subroutines Required

   Self contained program.

10. Sense Switch Settings

    None.

11. Timing

    Maximum: Not applicable.
    Average: Not applicable.
    Minimum: Not applicable.

12. Accuracy

    Not applicable.

13. Cautions to User

    See programming reference manual.

14. Equipment Configuration

    Model 33/35 A or B teletypes and 4096 words of memory.

15. References

    See programming reference manual.

METHOD

See program maintenance documents.

IDENTIFICATION

Title: Binary load dump program

Identification: BLD

Category: H10.00 - 1B04.620

Programmer: John H. Hathwell

Date: August 30, 1960

PURPOSE

To load and dump programs in standard binary format.

USE

1. Calling Sequence

   Call dump (X7434) with A = 1st address, B = last address and X = execution address (if X < O, then no execution address). Call load (X7630) with A < O to verify tape, A = O to load and return, A > O to load and execute.

2. Arguments or Parameters

   Subroutine entries shown above. Manual entry set A, B, and X and run at X7400 to punch bootstrap, X7404 to punch programs, X7600 to load programs.

3. Space Required

   377 octal.

4. Temporary Storage Required

   None.

5. Alarms or Printouts

   None.

6. Error Returns or Codes

   Punch: None.

   Load: A = load mode, B = 0 if good load or B = -1 if check sum error, X = last block address if check sum error or execution address if good load.

7. Error Stops

   Check sum error: IC = X7600, B = -1.

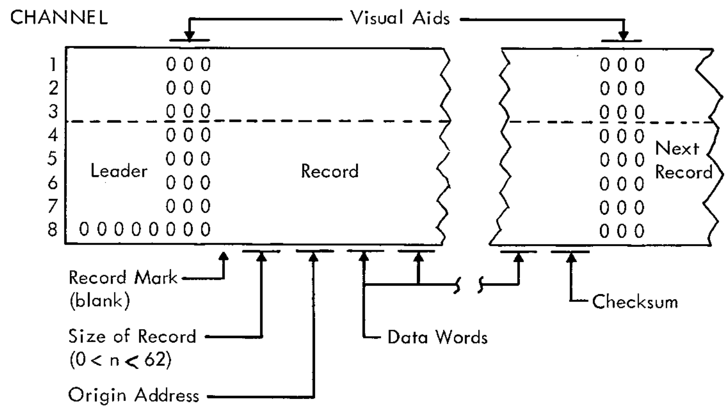
8. Input and Output Devices

   All standard peripheral devices.

9. Input and Output Formats or Tables

   Each word is punched three frames per word, six bits per frame, high order first. Channel 8 is not punched between visual aids. Channel 7 is the logical complement of channel 6. The checksum word is the exclusive OR of all preceding data words.

TAPE FORMAT



A record size of zero signals the end of a tape.

10. Sense Switch Settings

None.

11. Timing

Function of peripheral devices.

12. Accuracy

Not applicable.

13. Cautions to User

None.

14. Equipment Configuration

Minimum configuration of 4096 words and teletype.

15. References

See programming reference manual.

METHOD

Not applicable.

IDENTIFICATION

Title: Source tape correction program

Identification: COR

Category: C3, D3, H5

Programmer: J. L. Atwood

Date: August, 1965

PURPOSE

This program provides an easy means by which source program statements can be added to or deleted from a paper tape, and by which superfluous non-typing characters can be eradicated.

USE

1. Operational Procedures

   a. Insert the source paper tape in the model 33 teletype reader and prepare the reader and punch.

   b. RUN with the instruction counter set to symbolic location SENT + 1.

   c. When a halt occurs, proceed as follows:

      Type a new statement from the keyboard if a statement is to be added. Begin the statement with carriage return and line feed characters.

      Select sense switch 1 if the current input statement is to be deleted (i.e., read but not punched).

      Select sense switch 2 if the halt at the start of the next statement is to be by-passed.

      RUN

   d. Continue with step c until all source statements have been processed. Sense switch 2 may be changed at any time during the processing of a statement. A halt will occur at the end of the input tape regardless of sense switch 2 selection.

2. Arguments or Parameters

Not applicable.

3. Space Required

Approximately 110 words.

4. Temporary Storage Requirements

Not applicable.

5. Alarms or Printouts

A listing of the output paper tape is printed on the page printer.

6. Error Returns or Error Codes

Not applicable.

7. Error Stops

Not applicable.

8. Input and Output Devices

The model 33/35 teletype is used for both input and output.

9. Input and Output Formats

Each input statement is assumed to start with the carriage return or line feed character.

10. Sense Switch Settings

Sense switch 1, if selected, causes the current statement to be deleted.

Sense switch 2, if selected, causes the halt at the beginning of the next statement to be by-passed.

11. Timing

Not applicable.

12. Accuracy

Not applicable.

13. Cautions to Users

Each statement inserted should start with the carriage return and line feed characters.

Sense switch 1 should be changed only when the machine is halted.

14. Equipment Configuration

Minimum configuration with model 33 teletype.

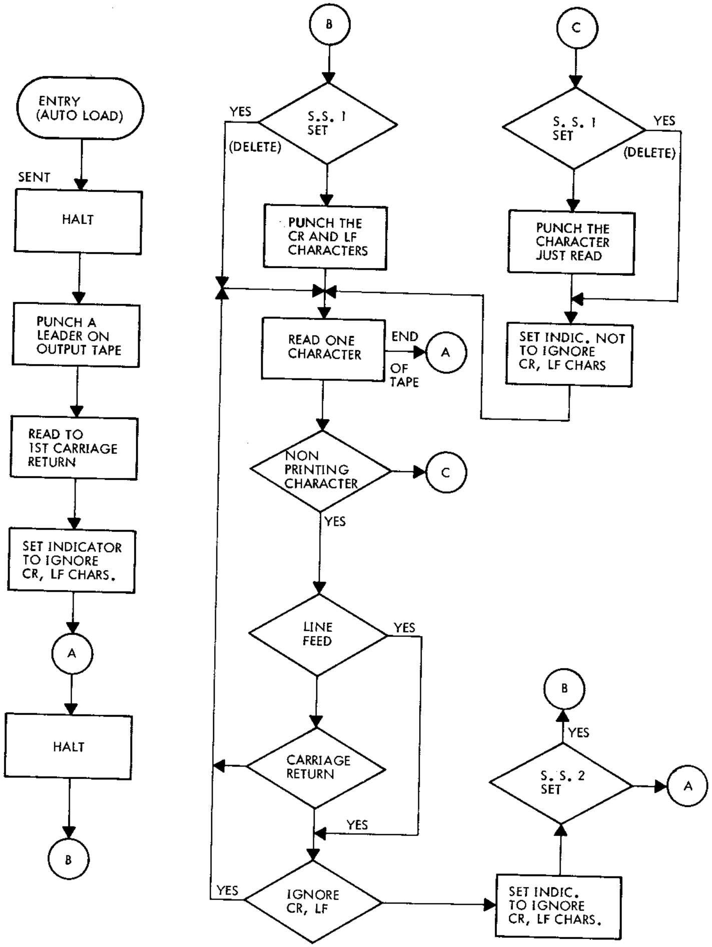
15. References

Not applicable.

METHOD

The carriage return and line feed characters are not duplicated, but are inserted by this program at the beginning of each statement duplicated.

All other non-typing characters are ignored when read.



SNDK = 0 IGNORE
$\neq$0 DON'T IGNORE

SECTION VI
EXECUTIVE ROUTINE

This section contains programmed routines separated into distinct packages. Each routine will follow the format described in section II, program description. As new routines are developed, they can be easily inserted into the proper section.

IDENTIFICATION

Title:	DATA 620/i assembler, mod. 1-F
Identification:	DAS 1-F
Category:	E10.00
Programmer:	John H. Hathwell
Date:	Septempber 1, 1966

PURPOSE

DAS 1-F provides translation from a mnemonic instruction language to DATA 620/i machine language. Input is source language instruction, consisting of mnemonic, symbolic instructions of two types: (1) symbolic machine instructions representing actual machine instructions, and (2) assembler instructions which control the location counters, define storage symbols, provide subroutine linkages, etc. Output is an octal machine language listing and/or object program (machine language) in absolute or relocatable.

DAS 1-F is a two-pass assembler. In the first pass, all location symbols are recognized and assigned locations. The second pass generates the listing and object outputs. The same source tape is used for both passes.

USE

1. Operating Procedures

DAS 1-F is stored on paper tape in binary load format, three frames per word.

After loading, the source is mounted. Pass A must be processed first if the source program contains any address symbols.

Pass A: object output off, SS1 on, SS2 off, SS3 off, (IC) = 0, (IR) = 0, RUN.

Pass B: SS1 off, to list SS2 on, for object SS3 on.

Run from system halt, registers will be correctly set.

2. Arguments or Parameters

None.

3. Space Required

4K memory:	6500 (8) words
8K memory:	11400 (8) words

4. Temporary Storage Requirements

Literals	4K	100 (8) words
	8K	400 (8) words
Pointers	4K	100 (8) words
	8K	400 (8) words
Symbol table (dictionary)		4 words per symbol defined

5. Alarms or Printouts

Not applicable.

6. Error Returns or Error Codes

As depicted in ref. (1). Approximately 25 distinct diagnostic codes are printed.

7. Error Stops

Synchronization error: HALT 0777, AR = BR = XR = 0777 (8). Press RUN to continue assembly.

8. Input and Output Devices

All standard input/output devices.

9. Input and Output Formats

Binary load/dump format.

10. Sense Switch Settings

See operational procedures.

11. Timing

Input/output limited.

12. Accuracy

Not applicable.

13. Cautions to Users

Do not attempt to restart assembly beyond the beginning of source tape.

14. Equipment Configuration

Minimum: DATA 620/i with 4096 words of memory and ASR-33 teletype.
Standard: DATA 620/i with 8192 words of memory and ASR-33 teletype.

15. References

(1) DATA 620/i programming reference manual
(2) Subroutine manual.

METHOD

The DAS 1-F assembler is a conventional two-pass assembler furnished with an extensive complement of assembly instructions as listed below.

Instructions for controlling multiple location counters:

BEGI(n), USE

Instructions controlling the current location counter:

ORG, LOC, BSS, BES

Instructions for generating data:

DATA, PZE, MZE

Instructions for calling and defining closed subroutines:

CALL, ENTRY, RETU(RN)

Conditional assembly instructions:

IFT, IFF, GOTO

Flag control instructions:

LIST, NLIST, PUNC, NPUN, SMRY, DETL, CONT,

NULL, SPAC, EJEC, MORE, END

Special controls:

DUP, READ

Instruction definition:

OPSY

Symbols defining controls:

EQU, SET, MIN, MAX

FORTRAN instructions:

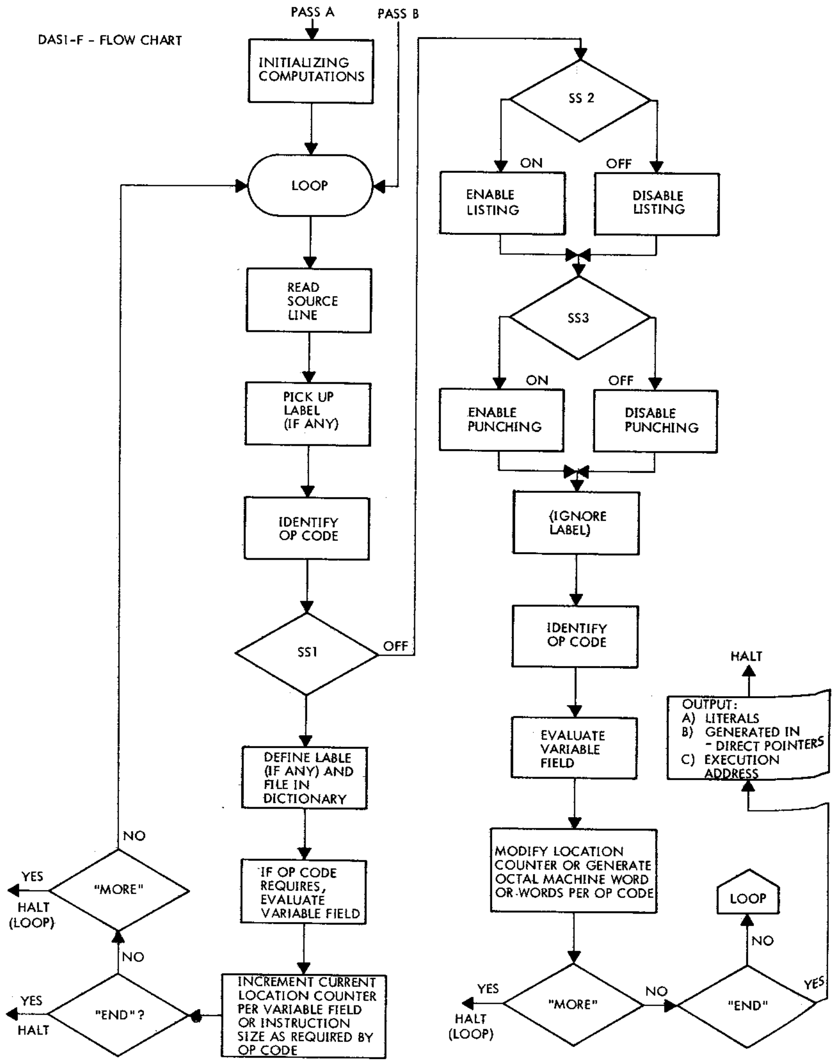
FORT, NAME, COMM, EXT

The 4K instruction set is a subset of the 8K instruction set listed above and includes the following:

1. ORG	5. BES	9. CALL	13. EQU
2. LOC	6. NULL	10. RETU(RN)	14. MZE
3. MORE	7. END	11. OPSY	15. PZE
4. BSS	8. DATA	12. SET	16. CONT

An overall review of DAS 1-F is furnished by the following flowchart.

DASI-F - FLOW CHART



INTERFACE REFERENCE

SECTION I

GENERAL DESCRIPTION

1.1 INTRODUCTION

The DATA 620/i computer is a high-speed, parallel binary computer. Its extensive instruction repertoire, flexible input/output, and modular packaging make it ideally suited for application as a general-purpose machine or as an on-line system component.

Its features include:

- Fast operation	1.8 microsecond memory cycle
- Large instruction repertoire	107 standard, with over 128 micro-instructions, plus 18 optional
- Expandable word length	16- or 18-bit configurations
- Modular memory	4096 words standard, 32,768 maximum
- Multiple addressing	Six, including direct, indirect, relative, index, immediate, and extended (optional)
- Flexible I/O	64 device addresses on standard I/O bus; optional interlaced input/output
- Extensive software	All programming and diagnostic aids required for efficient system use
- Modular packaging	Processor and 4K memory module occupy only 10-1/2 inches of rack space; additional memory module requires only 10-1/2 inches additional

The DATA 620/i is simple in design and is easy to program, operate, and maintain. As a system component, it is easily integrated with other equipment through the use of standard or special peripheral interface elements. The central processor and its associated power supplies and peripheral controllers all mount in standard 19-inch equipment cabinets and require no special cabling or air conditioning facilities.

1.2 PURPOSE OF THE MANUAL

This manual provides basic circuits and logic design, and timing information on the standard and optional input/output facilities of the DATA 620/i computer, plus design examples for seveal I/O functions. Using the information, the system designer may integrate the computer with special interfaces tailored to specific system requirement.

This manual also contains information on cabling, grounding, and installation procedures and thus serves as a basic document for system planning purposes.

While a detailed knowledge of the internal computer is not essential for successful interface design, it is recommended that the system designer have a general familiarity with the computer organization and operation. The available documents for the DATA 620/i are summarized in table 1-1. The reference manuals for the standard peripheral controllers will be particularly useful for design examples.

Table 1-1
DATA 620/i DOCUMENTS

VARIAN DATA MACHINES PUBLICATION NUMBER	TITLE
VDM-3000	System Reference Manual
VDM-3001	Interface Reference Manual
VDM-3002	Programming Reference Manual
VDM-3003	FORTRAN Manual
VDM-3004	Subroutine Manual
VDM-3005	Maintenance Manuals
VDM-3006	ASR-33 Teletype Controller Reference Manual
VDM-3007	Buffer Interlace Controller Reference Manual
VDM-3008	Magnetic Tape Controller Reference Manual
VDM-3009	600 LPM Line Printer Reference Manual
VDM-3010	300 LPM Line Printer Reference Manual
VDM-3011	Paper Tape System Controller Reference Manual
VDM-3013	Priority Interrupt Reference Manual

1.3 COMPUTER ORGANIZATION

The overall organization and basic information paths of the DATA 620/i computer are shown in figure 1-1. The basic system is composed of the following functional elements: memory section, control section, arithmetic/logic section, operational register section, and input/output section. An optional input/output facility, direct memory access is also available.

1.3.1 Memory Section

Memory modules are slaved to the central processor, which contains the address and data registers for all modules. Minimum memory size is 4096 words. The memory may be field expanded by the addition of pre-wired memory modules. Interconnecting wiring is accomplished by the installation of tagged wires, terminated with slip-on terminals.

1.3.2 Control Section

The control section decodes the program instructions into timing and control signals for the entire machines. There are 107 standard instructions decoded; an additional 18 instructions may be supplied as options. Over 128 microcoded instructions may be derived from the standard instruction set.

1.3.3 Arithmetic/Logic Section

This section contains the gating elements required to perform all programmed arithmetic and logic operations. It is also used for internal control operations such as instruction and operand address modification.

1.3.4 Operational Register Section

Operational registers include the A, B, X, and P registers. A and B form a double-length register for arithmetic and logical operations. The B register may also be used for indexed addressing. The X register is a full 16-bit hardware index register. Indexed addressing using B or X requires no additional time for execution of the instruction. Registers A and B may also be used for direct input/output transfers. The instruction counter, P, holds the memory address of the instruction being executed by the control sections. The S bus provides routing of these registers to the arithmetic unit.

1.3.5 Input/Output Section

This section provides transmission of control and data signals to and from peripheral devices attached to the I/O cable. A total of 64 peripheral device addresses are available. External program sense and interrupt functions are also transferred to and from the control section through the I/O section. Data transfers may be single-word (program controlled) or block (using the optional buffer interlace controller).

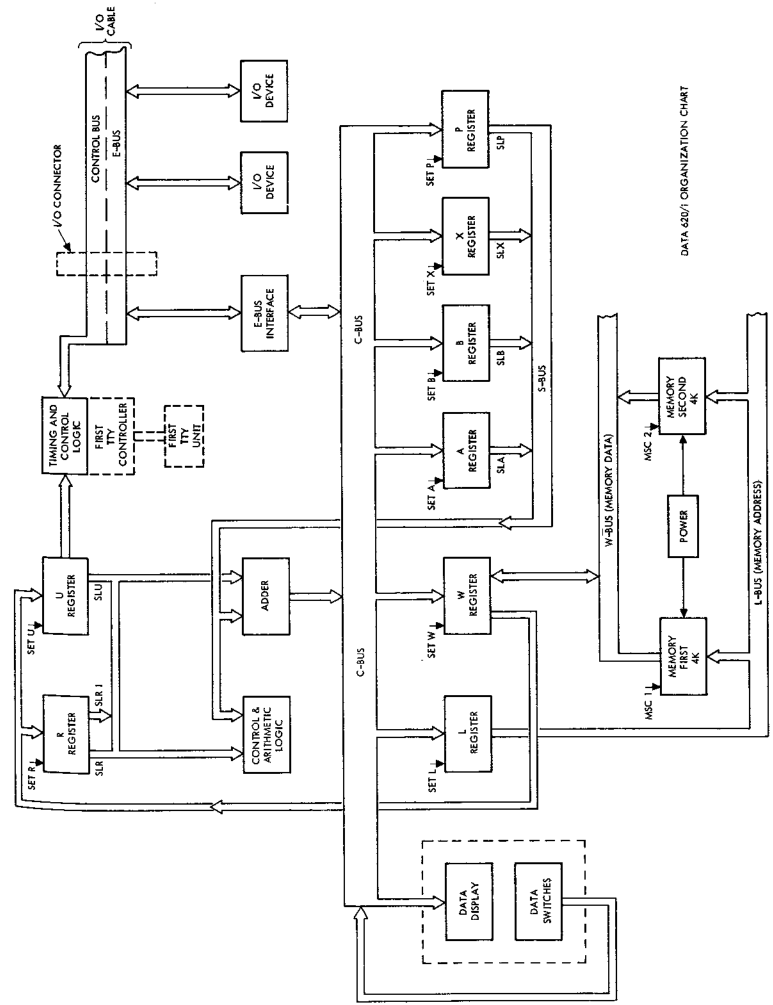


Figure 1-1. DATA 620/i Organization.

SECTION II

DATA 620/i STANDARD INPUT/OUTPUT SYSTEM

2.1 ORGANIZATION

As shown in figure 1-1, the I/O section of the computer communicates with the operational registers and the memory through the C bus. Data and control signals are transmitted to and from external peripheral devices through the I/O cable.

2.1.1 Overall Operation

The overall organization of the DATA 620/i I/O system, including a typical set of peripheral devices is shown in figure 2-1. Standard or special peripheral devices are in parallel on the I/O bus. Any number of logical devices, up to a total of 64, may be added. The following types of information transfers between the central processor and the external devices through the I/O bus may be executed:

External control. An external control code is transmitted under program control from the central processor to a device.

Program sense. The central processor can sense the status of a selected external line under program control.

Single word transfer to/from the A and B registers. A single word may be transferred to or from the A and B registers under program control.

Single word transfer to/from memory. A single word may be transferred to or from any memory location under program control.

Program interrupt. An external device may force the central processor to execute an instruction in a specified location in the memory.

Buffer interlace controller (BIC) transfer to/from memory. Blocks of words may be transferred to or from sequential memory locations under control of an optional buffer interlace controller. Devices controlled by the BIC may also be operated under program control (single word transfers).

Interlace data transfers. Single words may be transferred to/from memory by the control signals available on the I/O cable. Buffer interlace controllers use the lines for performing interlaced data transfers.

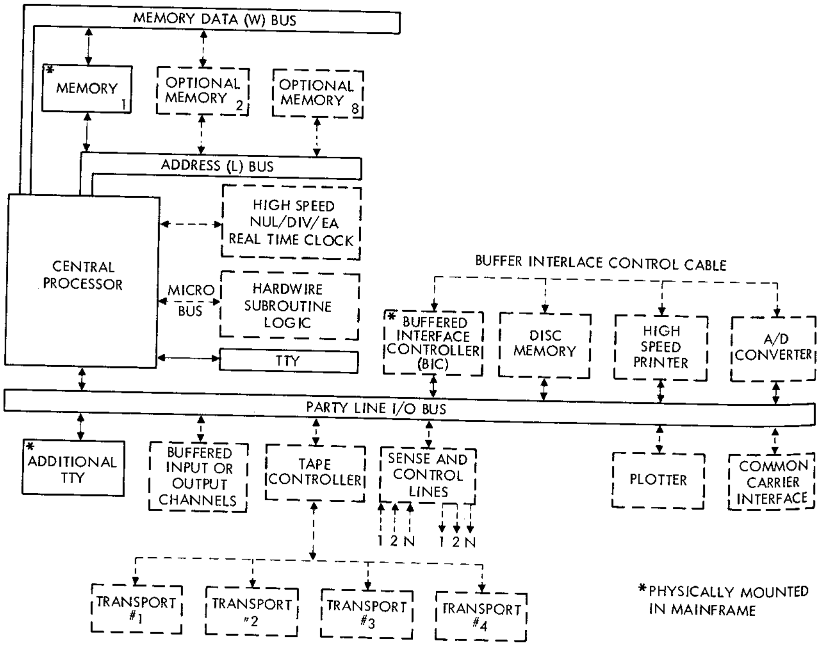


Figure 2-1. DATA 620/i I/O System Organization.

2.1.2 I/O Cable

A typical functional organization of peripheral devices on the I/O cable is shown in figure 2-2. The I/O cable consists of the E bus plus a set of control lines. The E bus contains 16 (or 18) pairs of bi-directional lines which transmit control codes, addresses, and data between the central processor and the peripheral devices connected in parallel to the cable. The 5 I/O control lines transmit timing signals to and from the central processor to synchronize the information transfers over the E bus.

Information transfers with the DATA 620/i are synchronized by peripheral controllers; these controllers may, in turn control one or more peripheral devices. The central processor can communicate directly with all peripheral controllers under program control. It may determine when a device is ready to send or receive information by sensing associated sense lines, or it may be notified by means of a program interrupt. All standard peripheral controllers contain the necessary sense and external control functions for proper operation.

Priority interrupt and sense line modules are available for use for special system interfacing.

When block transfers of data independent of program control are required (such as from tapes, drums, commutators etc.), the buffer interlace controller may be provided. This element contains hardware registers which automatically generate the proper memory addresses for successive data transfers with the DATA 620/i memory and a device through its controller.

2.1.3 Input/Output Operations

All I/O operations are either one or two phase; sense and external control are single-phase while data transfers are two-phase. Each phase is terminated with a control pulse. During information transfers over the I/O bus, the E lines may carry control codes, addresses, or data, depending on which type of operation is being performed. The control signals defining the type of operation are listed in table 2-1. Table 2-2 shows optional interrupt control signal information. Tables 2-3 and 2-4 summarize the information carried on the E bus for the specified operations. The timing signals present on the I/O control lines during each operation are also indicated.

When a control is on the E bus (first phase), lines EB11-EB15 carry a control signal which defines the operation. The control codes transmitted over E bus, are summarized in table 2-5. The function ready (FRYX-I) pulse is generated to indicate that a control code is on the E bus.

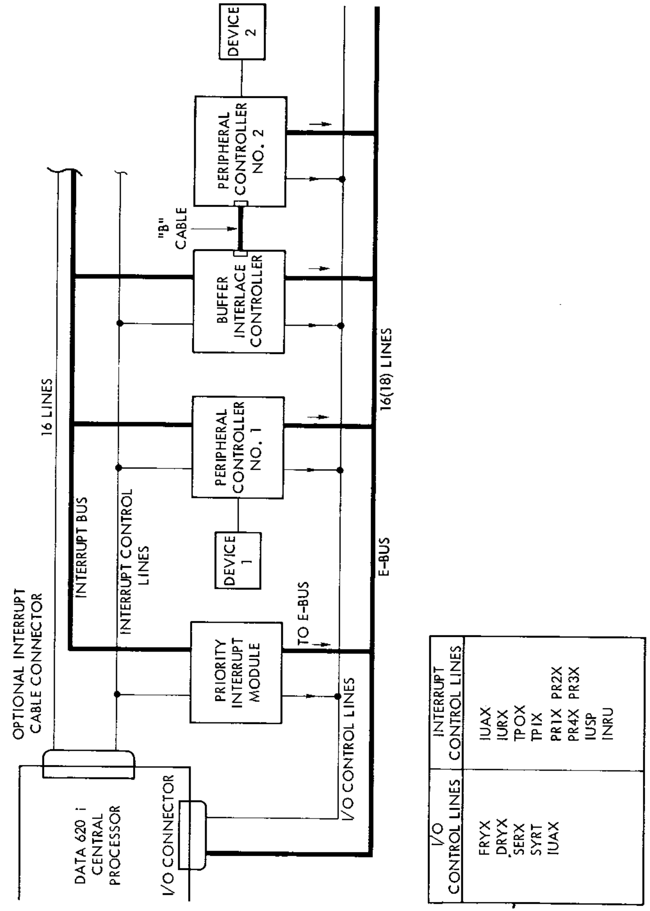


Figure 2-2. DATA 620/i I/O System, Functional Diagram.

Table 2-1
I/O CONTROL LINE SIGNALS

CONTROL LINE	SYMBOL	FUNCTION
Function Ready	FRYX-I	Indicates that the E bus contains address information. The type of address depends upon the state of IUAX-I.
Data Ready	DRYX-I	Indicates that the E bus contains data.
Sense Response	SERX-I	Indicates logical state of line designated by sense line address on E bus.
Interrupt Acknowledge	IUAX-I	Indicates that external interrupt demand is being acknowledged. Address is placed on E bus and removed at FRYX-I.
System Reset	SYRT-I	Line which becomes true when the SYSTEM RESET button on the control console is pressed. Used to initialize each controller connected to the I/O cable.

2.1.4 Input/Output Section Logic and Connector

The logical organization of the DATA 620/i I/O section and layout of the standard I/O connector are illustrated in figure 2-3 and detailed in table 2-6. E bus outputs from the computer are transmitted by a set of line driver circuits; these signals are gated through drivers by the internally generated E bus drive signal (EBDX+). E bus inputs to the computer are gated through the E bus receivers by the internally generated E bus receive signal (EBRX+).

The computer I/O connector has a termination "shoe" inserted. This "shoe" contains terminating resistors to +3 volts. When adding an additional device to the system, the termination "shoe" is removed and installed on the second connector of the added device, with the interconnecting cable in its place.

Table 2-2
INTERRUPT CONTROL LINE SIGNALS

CONTROL LINE	SYMBOL	FUNCTION
Interrupt Request	IURX	Indicates a demand for the interrupt module to force program to execute one instruction at the location specified by address on E bus. This address will be placed on the E bus when IUAX becomes true.
Interrupt Acknowledge	IUAX	Indicates that external interrupt demand is being acknowledged. Address is placed on E bus and removed at FRYX.
Trap Output	TPØX	Inidcates that a buffer interlace controller (or equivalent) is requesting a data transfer from memory.
Trap Input	TPIX	Indicates that a buffer interlace controller (or equivalent) is requesting a data transfer to memory.
Interrupt Clock	IUCX	A 1.1 MHz clock. Clock is off when IUAX is true.
Priority Out	PRIX	Priority line used with interrupt and buffer interlace controller modules for priority determination.
Priority In	PR4X	Priority line returned to computer to permit console interrupt.
Priority 2 and 3	PR(N)X	Intermediate priority lines that are used on the I/O bus allowing flexible priority assignments.
Interrupt Jump	IUJP	Indicates that a jump-and-mark intruction is being executed for an interrupt request.
Increment and Replace Interrupt	INRU	Echo pulse generated by the processor when the instruction "increment memory and replace" is executed under interrupt control and fit 14 of the memory word being.

Table 2-3
INTERRUPT CABLE SIGNAL MATRIX

<table>
<tr><td colspan="2" rowspan="3"></td><td colspan="3">OPERATION</td></tr>
<tr><td colspan="2">TRAP SEQUENCE
(BUFFER INTERLACE CONTROL)</td><td>INTERRUPT
SEQUENCE</td></tr>
<tr><td colspan="2">TPOX-I or TPIX-I</td><td rowspan="2">IURX-I
IUAX-I
(PHASE 1)</td></tr>
<tr><td></td><td>CONTROL
LINES</td><td>IUAX-I, FRYX-I
(PHASE 1)</td><td>IUAX-I, DRYX-I
(PHASE 2)</td></tr>
<tr><td rowspan="8">E Bus Meaning</td><td>EB00-I
to
EB05</td><td rowspan="8">Memory
Address
In</td><td rowspan="8">Data
In
or
Out</td><td rowspan="8">Use lines
00-15 for
interrupt
Location
by pairs.</td></tr>
<tr><td>EB06-I
to
EB08-I</td></tr>
<tr><td>EB09-I
EB10-I</td></tr>
<tr><td>EB11-I</td></tr>
<tr><td>EB12-I</td></tr>
<tr><td>EB13-I</td></tr>
<tr><td>EB14-I</td></tr>
<tr><td>EB15-I</td></tr>
</table>

Table 2-4
I/O CABLE SIGNAL MATRIX

CONTROL LINES (E Bus Meaning)	OPERATION: EXTERNAL CONTROL	OPERATION: SENSE	OPERATION: DATA TRANSFER (SINGLE WORD I/O)	
	FRYX-I* (PHASE 1)	SERX-I* (PHASE 1)	FRYX-I* (PHASE 1)	DRYX-I (PHASE 2)
EB00-I to EB05	Device Address	Device Address	Device Address	
EB06-I to EB08-I	Function Code	Function Code	Not Used	
EB09-I EB10-I	Not Used	Not Used	Not Used	Data being Trans-fered In or Out
EB11-I	External Control Command			
EB12-I		Sense Command		
EB13-I			Data Transfer In	
EB14-I			Data Transfer Out	
EB15-I				

*IUAX Interlock – used in the address decoding term.

Table 2-5
SUMMARY E BUS SIGNALS

OPERATION	EB15-I	EB14-I	EB13-I	EB12-I	EB11-I	EB10-I, EB09-I	EB08-I	EB07-I	EB06-I	EB05-I to EB00-I
Output from Memory		1	0	0	0	Unused	0	0	0	Device Address $(00\text{-}63_{10})$
Output from A Register							Unused Note 1	0	1	
Output from B Register							Unused	1	0	
Input to Memory		0	1	0	0	Unused	0	0	0	Device Address $(00\text{-}63_{10})$
Input to A Register							Note 1	0	1	
Input to B Register							Note 1	1	0	
Sense State of External Device		0	0	1	0	Unused	Function Code			Device Address $(00\text{-}63_{10})$
Send function code to External Device		0	0	0	1	Unused	Function Code			Device Address $(00\text{-}63_{10})$

Note: If EB08-I is true, selected register in computer is cleared.
If EB08-I is false, selected register in computer is not cleared.
Bits EB06- to EB08- generally ignored by I/O controller, during input or output commands.

75 PIN CONNECTOR - CONNECTOR PIN ASSIGNMENTS

Pin	Function	To	
1	EB00-I	12	108
2	R		95
3	EB01-I		102
4	R		93
5	EB02-I		76
7	R		86
8	EB03-I		72
10	R		44
11	EB09-I		30
12	R		42
13	EB05-I		26
14	R	12	8
15	EB06-I	13	108
16	R		95
17	EB07-I		102
18	R		93
20	EB08-I		76
21	R		72
22	EB09-I		72
23	R		44
24	EB10-I		30
25	R		42
26	EB11-J		26
27	R	13	08
28	EB12-I	14	108
29	R		95
30	EB13-I		102
31	R		93
32	EB14-I	14	76

Pin	Function	To	
33	R	14	86
34	EB15-I		72
35	R		44
36	EB16-I		30
37	R		42
38	EB17-I		26
39	R	14	8
40	FRYX-I	19	05
41	R		01
42	DRYX-I		11
43	R	19	01
44	SERX-I	16	98
45	R	16	122
46	TPIX-I		
47	R		
48	TPØX-I		
49	R		
50	PR1X-I		
51	R		
52	PR2X-I		
53	R		
54	PR3X-I		
55	R		
56	PR4X-I		
57	R		
58	SYRT-I	24 -	06
59	R	24 -	01
60	IUAX-I		
62	R		

Pin	Function	To	
63	IUCX-I		
64	R		
65	IURX-I		
66	R		
67	IUJX-I		
70	R		
71	IØCL-I		
72	R		
73			
74			
75			
76			
77			
78			
79			
80	+3VDC	X16-0+9	
82	GND	X16-001	

NOTES:

TWISTED PAIR

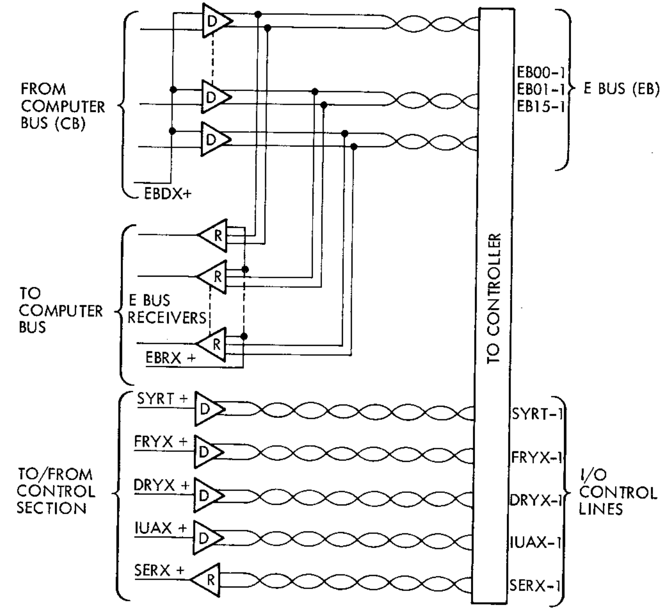


Figure 2-3. DATA 620/i I/O Section and Standard Connections.

2.1.5 Logic Levels

Logic levels for Micro-VersaLOGIC circuits are nominally 0 volt for a logic "0" and +5 volts for logic "1". Over the I/O cable, however, the sense of the logic signals is inverted and the voltage is changed. That is, binary "1's" are transmitted over the E bus at the 0-volt level and binary "0's" are transmitted at the +3-volt level Control lines rest at the +3-volt level; a control pulse is defined by the signal level dropping to 0 volt for the prescribed time interval, and then returning to the +3-volt level. The standard line receivers convert the I/O cable signals to 0 and +5 volts while the line drivers convert the 0- and +5-volt signals to the I/O cable signals. One line of the twisted pair is terminated at each end of 180 ohms to +3 volts, with the line grounded. The line driver acts as a switch across the pair to bring the potential difference between the lines to zero (indicating a logic "1"). When the driver is turned off, the voltage returns to +3 volts, (indicating a logic "0"). The drivers are capable of supplying 60 ma of current. The receiver input impedance is approximately 3.7K ohms. Up to 10 receivers may be added to any twisted pair, and up to 20 drivers may drive any twisted pair. Figure 2-4 shows one signal.

2.2 PROGRAM CONTROL FUNCTIONS

Interfacing functions fall into two major categories: programmed operations, and automatic operations. The programmed operations are: external control (single bit out), sense operations (testing a single bit), data transfer in (full word inputs), and data transfer out (full word outputs). The following paragraphs describe the programmed operations and examples of their use. The party line adapter is a special card for use in interfacing the programmed operations.

2.2.1 I/O Cable Adapter Card

The I/O cable adapter card is a standard Micro-VersaLOGIC module (I/O-701) designed to facilitate interfacing with the DATA 620/i I/O cable (see figure 2-5). Subsequent paragraphs show typical examples illustrating the use of the I/O adapter. The organization of this card is such that many types of I/O interfaces may be simplified by its use. The address detection gates are used for forming the address; this also incorporates the IUAX-I signal for address lock-out during trap and interrupt sequences. The two flip-flops are used to implement the two-phase technique for I/O transfers (i.e., remember whether data is being transferred in or out). In some cases, one of the two flip-flops is used to implement a buffer ready function. The sense response driver (connected directly to the SERX-I line) has a logic inverter to allow direct ORing of many sense functions. The power driver is multipurpose.

The various uses of the I/O cable adapter card are shown in paragraphs 2.2.2 through 2.2.5.

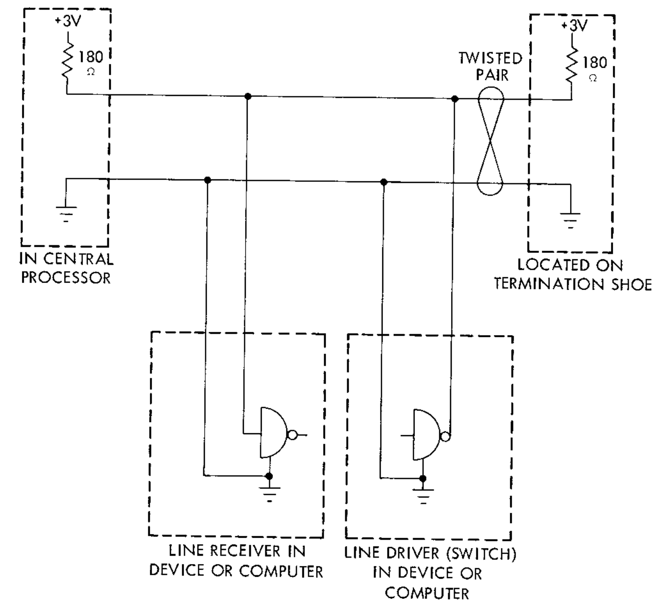


Figure 2-4. Typical Line Location on I/O Bus.

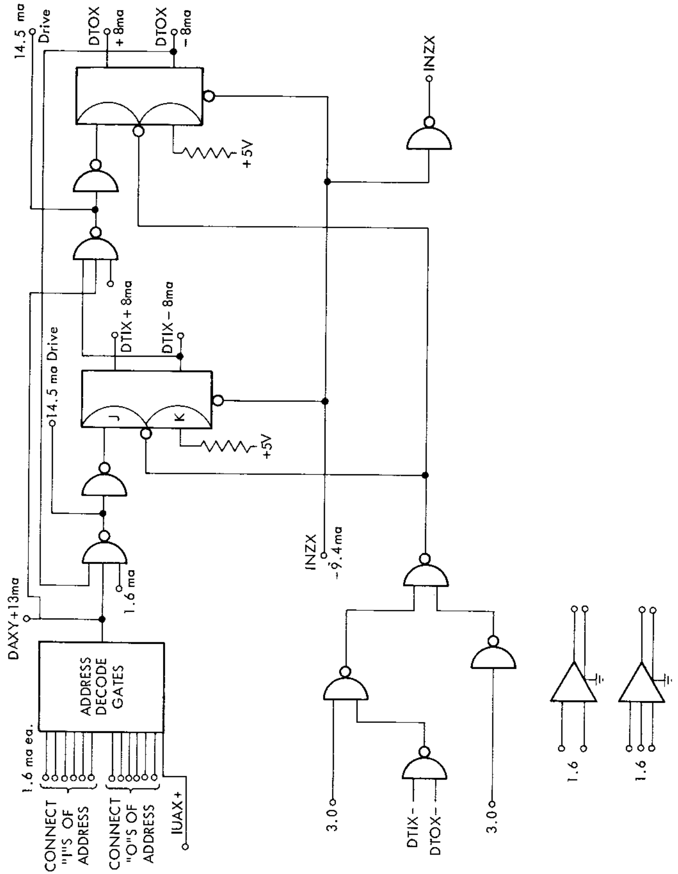


Figure 2-5. I/C Cable Adapter Card 10-701, Functional Organization.

2.2.2 External Control Operation

External control operations are single-phase operations. The external control instructions (EXC XYY, where YY contains the device address and X contains the function code) transmits a function code and a device address on the E bus for 900 nanoseconds (figure 2-6). Functions EB00 through EB05 contain a device address, and bits EB06-08 indicate the particular function code for that device. EB11 is true indicating that an external control function is being performed (see table 2-3). The pulse FRYX+ is used with the address to form a 450-nsec pulse for setting and resetting flip-flops. The address overlaps FRYX+ by 100 nsec to allow for logic delays in forming the pulse signal in the power drivers.

An example of implementing eight external control lines is shown in figure 2-7. This example requires four Micro-VersaLOGIC cards. As shown in figure 2-7, only the meaningful I/O signals need to used to form the external control function. The output of the select gate (EB06-08 describes one of eight) is a 450-nanosecond pulse (GND true).

2.2.3 Program Sense Function

Program sense functions are single-phase operations. The sense instruction is a two-word instruction. The first word in the sense instruction contains the function code and device address which addresses a particular external sense function. The second word is the conditional jump address. The sense instruction transmits the function code of the E bus for 1350 nanoseconds (see figure 2-8). Lines EB00-EB05 contain the device address, lines EB06-EB08 dictate the particular function to be sensed, and EB12 is true indicating a sense command. The EB12 lines need not be used in forming a sense response command because the computer will not respond to the SERX-I line unless a sense command is being executed. The function (address IUAX-I) can be directly used to enable a sense line driver. The user has the option of using the EB12 line for any case where he must know if a function is being sensed. The FRYX-I signal is normally not used for a sense response command, but is furnihsed for the user that desires a clocking pulse while performing a sense function. The SERX-I line is the return line to the computer with all sense line drivers connected to this line. The SERX-I line must be driven within 600 nanoseconds after time T_0 (see figure 2-8), if the computer is to recognize a "sense condition met".

An example of sense function decoding is shown in figure 2-9. This example illustrates the logic required to implement either sense functions. The line receivers interface with the I/O cable signals shown. Lines EB00-EB05 plus the signal IUAX+ are used at the address detection gate to form the enable for all sense lines. Lines EB06-EB08 are decoded into the six combinations shown, with the final decoding provided on the eight NAND gates with the corresponding sense lines. The AND/OR function is formed by attaching the NAND outputs and inverting. This function enables the line driver circuit (the inverter and line driver are located on the adapter card).

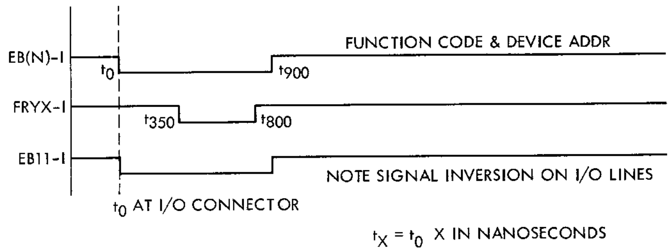


Figure 2-6. External Control Timing.

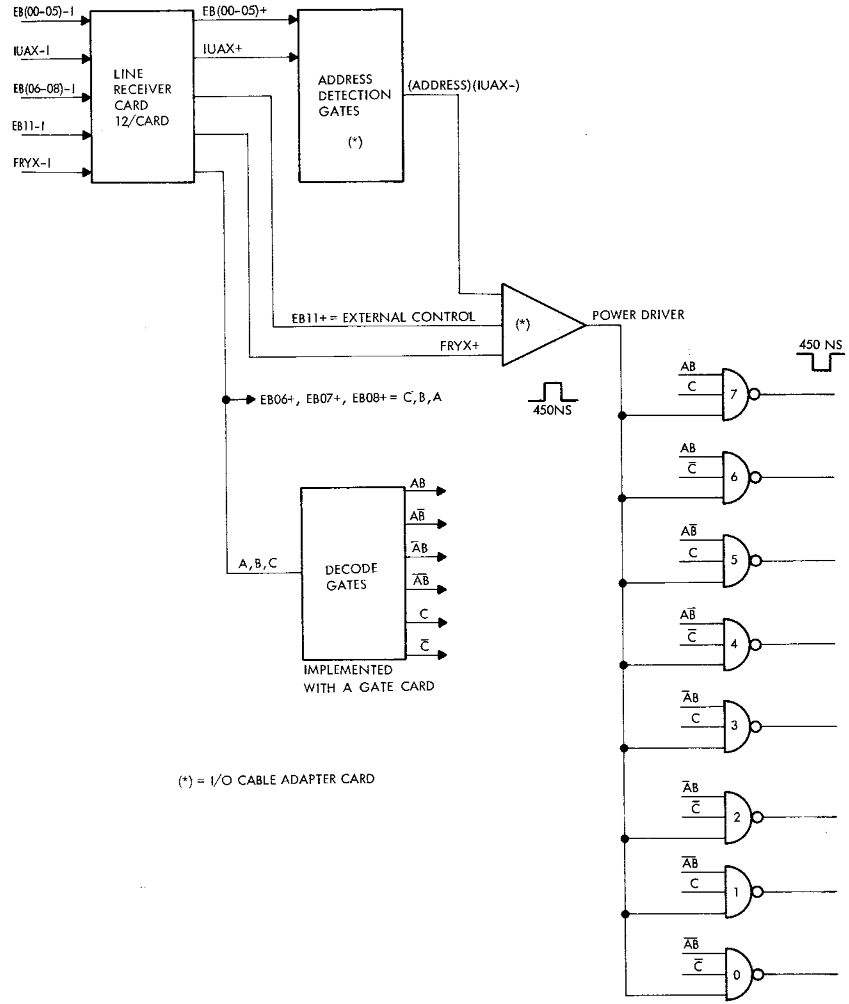
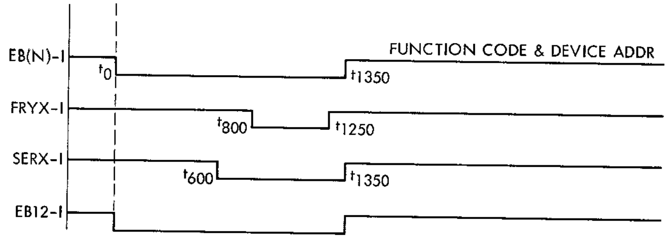


Figure 2-7. Example of External Control of Eight EXC Lines.



$t_X = t_0 + X$ IN NANOSECONDS

SERX-MUST BE ON (IF RESPONSE IS TRUE) t_{600},
NORMALLY OFF t_{1350}, MUST BE OFF t_{1950}

NOTE SIGNAL INVERSION ON I/O LINES

Figure 2-8. Sense Response Timing.

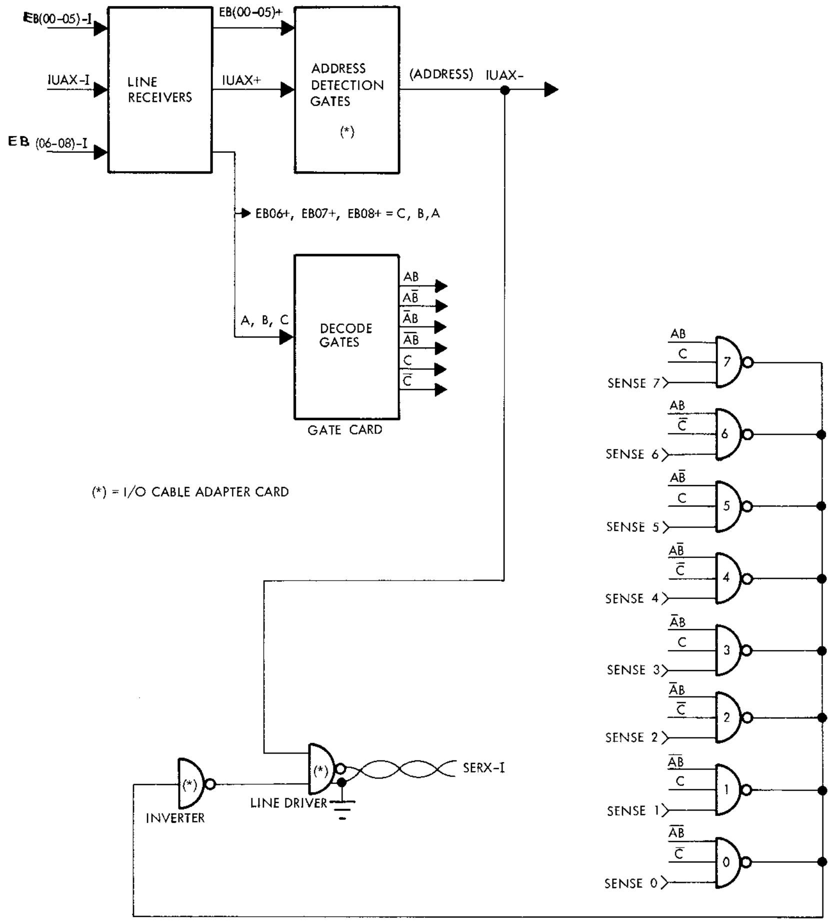


Figure 2-9. Example of Sensing Eight Sense Lines.

2.2.4 Data Transfer In Operation

Data transfer is in a two-phase I/O operation (both phases are completed during one instruction). The device address is transmitted during the first phase. During the second phase, data is placed on the E bus by the addressed I/O device. Data is transferred into the computer by one of the data input instructions, either to one of the computer registers or directly into the memory. The first-phase timing is similar to the first phase of other I/O functions. EB13 is true to indicate a data transfer in function (lines EB00-EB05 contain the device address).

Since the E bus is time-shared, a flip-flop (in the selected device) is used to remember that the addressed device is to place data on the bus during the second phase. This flip-flop, data transfer in (DTIX+), is set at the trailing edge of FRYX+ (with the proper enabling conditions), reset with the trailing edge of DRYX+, thus enabling data onto the E bus. The timing of the data transfer in operation is shown in figure 2-10. As indicated, the selected data must be enabled onto the E bus no later than 700 nanoseconds after the trailing edge of the pulse FRYX+.

An example of the data transfer in operation, shown in figure 2-11 illustrates the standard gated input channel.

When data is present on the 18 customer-driven data lines, the customer causes the data present line to go to a +5 V. When the computer addresses a sense command to the device, the power driver causes the sense response (SERX-I) line to be grounded, indicating a favorable response.

The computer then addresses a data transfer in command to the device. The DTIX flip-flop is set by function ready (FRYX+) during the first phase of the input command.

DTIX gates the customers' data onto the E bus during the second phase of the input command.

The trailing edge of the 450-nanosecond Data ready (DRYX+) pulse resets DTIX, removing the data from the bus. The term DTIX+ o DRYX+ is supplied to the customer as a data accepted pulse, after which the customer may remove data from the lines.

2.2.5 Data Transfer Out Operation

Data is transferred from the computer to an external device by one of the data output instructions. Data from the computer can originate from one of the computer registers or directly from the memory. The data transfer out is a two-phase operation where the first phase outputs the function code (EB00-EB05 = address and EB14 = data output). This phase is terminated and the selected device strobes this information with the pulse DRYX+. As shown in figure 2-12, the computer removes the data 100 nanoseconds after the DRYX+ pulse. The overlap of 100 nanoseconds allows the user to form a

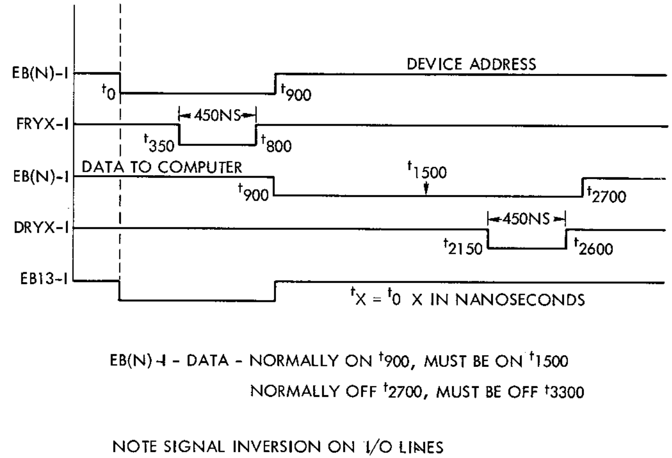


Figure 2-10. Timing of Data Transfer In.

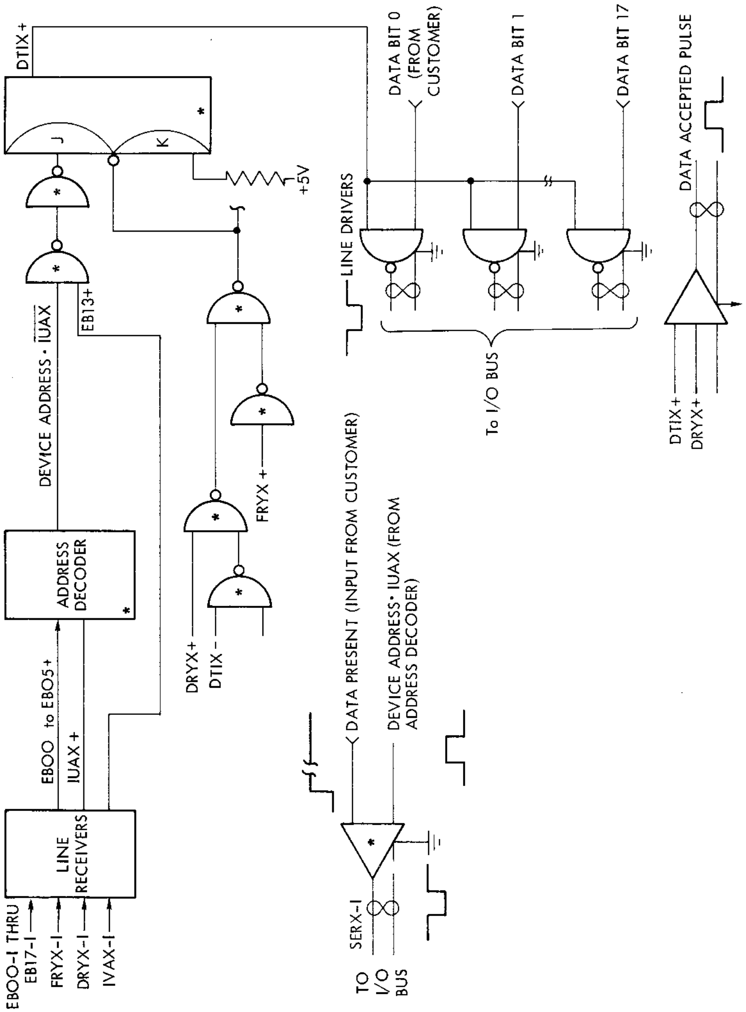


Figure 2-11. Example of Gated Data Transfer In.

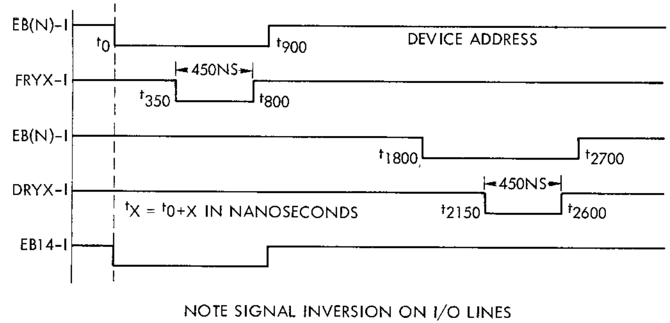


Figure 2-12. Timing of Data Transfer Out.

register-set pulse with a power driver and strobe EB00 - EB(N) information into an external flip-flop register. Since the address code is not on the E bus during the second phase, a flip-flop is used to store the device selection. This flip-flop is called DTØX+ (data transfer out) and is used to enable the DRYX+ pulse to strobe the E bus data into the register. The DTØ flip-flop is set during the first phase of the I/O instructions with FRYX+, and is reset during the second phase with DRYX+.

Figure 2-13 shows an example of a data transfer out operation with a standard gated output channel. The external device must request data with a request level (output ready). This signal is connected to the sense return line and may be sensed by the computer at any time.

2.3 AUTOMATIC CONTROLLED FUNCTIONS (optional)

Automatic controlled functions, especially interrupts and traps, can demand the computer system to perform a function that is independent of a particular instruction being executed. The program-controlled functions of paragraph 2.2 are all executed under control of DATA 620/i instructions.

2.3.1 Priority Lines and Interrupt Clock

The devices that connect to the I/O and interrupt cable and perform demand-type functions must first establish a priority to resolve two or more simultaneous demands to the computer. The priorities of the devices are determined every 1.8 microseconds and are clocked with the interrupt clock IUCX-I (a 1.1-MHz signal). The computer sends a priority out signal (PR1X-I, see table 2-2) when a device may have priority, and receives a priority in signal (PR4X-I) when no device is demanding computer intervention.

The intermediate priority lines (PR2X-I, PR3X-I, and PR4X, see table 2-2) are used to allow the designer to assign priorities to units not physically adjacent. The only requirements in priority logic are that the chain not be broken unless the demand device wants to interrupt or trap the computer.

If the PR1X-I signal is true, requests will be accepted from a device. This signal is false only when a power failure has been detected and the power fail interrupt is in process; during that time, all trap requests from the devices on the interrupt bus are ignored by the DATA 620/i.

The priority assignment among multiple devices on the priority bus is made by inhibiting a trap request from one unit when a request from a higher-priority unit is on. Thus, each device has priority logic which receives a priority input which, when true, indicates that it may generate a request. The output of this priority logic is set false when the device is generating a request, indicating that no unit of lower priority may generate a trap or interrupt request.

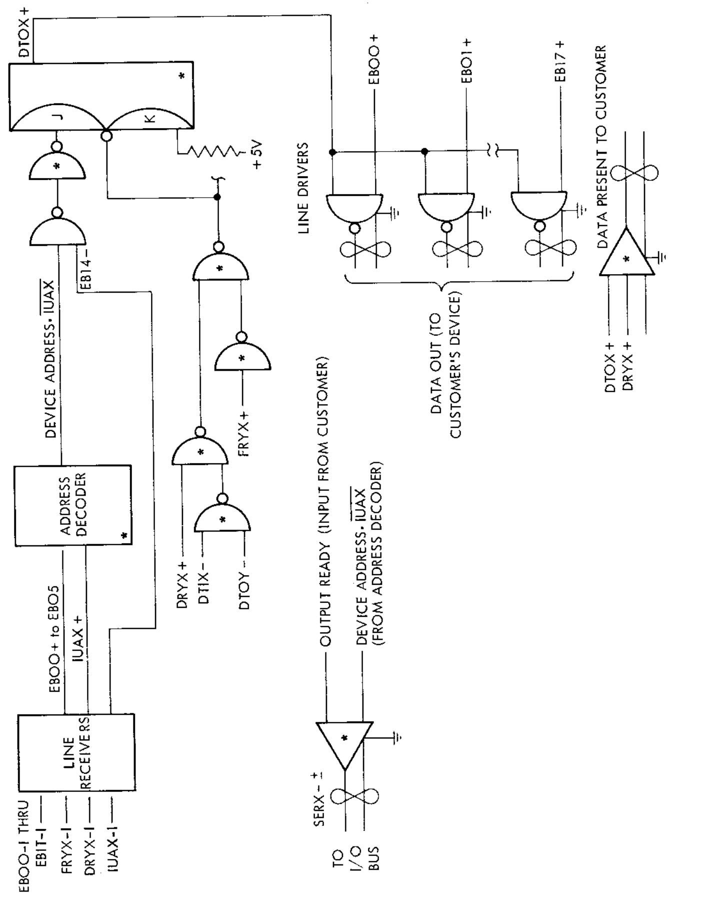


Figure 2-13. Example of Gated Data Transfer Out.

The simplest assignment of priorities is to let the physical position on the interrupt cable determine the priority. This is illustrated in figure 2-14. The priority output (PRIX-I) from the central processor serves as the input to the highest-priority logic, its output is the input to the second, and so on. When the highest-priority unit generates a trap request, all lower priority units are inhibited from generating a trap or interrupt request.

Where physical location on the interrupt cable does not correspond to the priority assignment, an arrangement such as illustrated in figure 2-15 is used. The priority of each device may be set up as desired and the priorities may be reassigned at any time by a simple change of jumpers.

The interrupt clock (IUCX-I) line is used by all devices that will request either an interrupt or a trap from the computer. All requests should be turned on at the IUC time so that multiple requests have time to settle the priority chain, and lower-priority requests may remove their signals before the interrupt acknowledge signal (IUAX-I).

2.3.2 Computer Interrupts

The following paragraphs describe the pholosphy for requesting and acknowledging interrupts. The interrupt module, model 620/i-27, is implemented using these control lines.*

As shown in figure 2-16, the signals used are interrupt request (IURX-I), interrupt acknowledge (IAX-1), and the E bus for sending the interrupt address to the computer. When an interrupt device wants to execute an interrupt, the device places an interrupt request signal on the interrupt cable, if the priority line coming into the device is true. The device must also set false the priority signal for all downstream requesting devices. After the completion of the instruction being executed, the computer will respond with an interrupt acknowledge (IUAX-I).

As shown in figure 2-16, the IURX-I signal will be true for a variable period of time until the IUAX-I signal. This time will vary depending on the instruction being executed. The device must have the interrupt address on the E bus 600 nanoseconds after IUAX-I becomes true, and must remove the address and IURX-I signals within 150 nanoseconds after IUAX-I goes false.

2.3.3 Interlaced Data Transfers

The following paragraphs describe the pholosophy for performing data (full word) transfers directly to and from the memory connected with the computer. The buffer interlace

*The reader should consult the interrupt module manual for a detailed description of the operation and interface of the interrupt module.

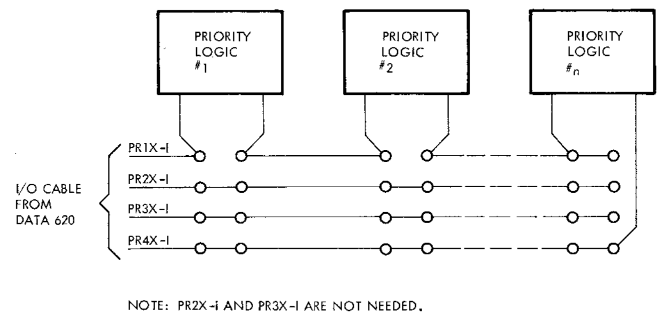


Figure 2-14. Priority Assignment by Physical Order on the I/O Bus.

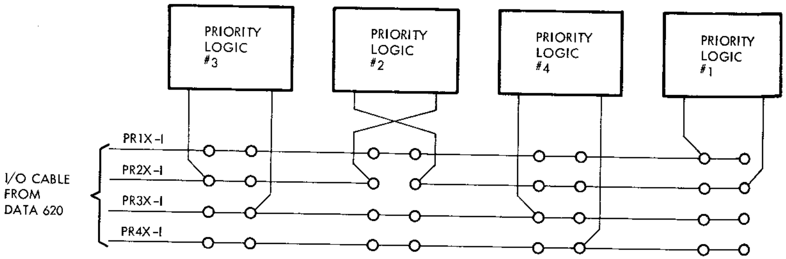
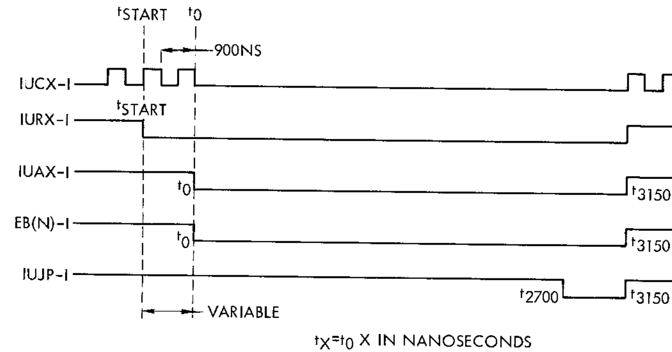


Figure 2-15. Priority Assignment by Jumpers.



IUCX-I - REMOVED AT t_0, RETURNS t_{3150}

IURX-I - NORMALLY REMOVED AT t_{3150} MUST BE t_{3300}

IUAX-I - NORMALLY REMOVED AT t_{3150}, MUST BE t_{3300}

EB(N)-I - ADDRESS NORMALLY ON t_0, MUST BE ON t_{600}

IJUP-I - PRESENT IF JUMP-AND-MARK INSTRUCTION

NOTE SIGNAL INVERSION ON I/O LINES

Figure 2-16. Timing of Interrupt Sequence.

control (BIC, model 620/i-15) is implemented using the following technique. (The interested user should consult the BIC manual for its use and interfacing requirements).

Basically, the trap (interlace) sequence is a three-phase operation: request, address, and data. First, the device requests a trap into or out of memory (with a TPIX-I or TPØX-I). Second, the computer acknowledges with an IUAX-I and the device places the address of the desired memory location on the E bus, and third, after the computer responds with a FRYX-I, the data is placed on the E bus (either from the device or from the computer). The sequence ends with a DRYX-I pulse that strobes the data into or out of the computer and all signals are removed from the bus (see figure 2-17).

2.4 MISCELLANEOUS SIGNALS

2.4.1 System Reset (SYRT-I)

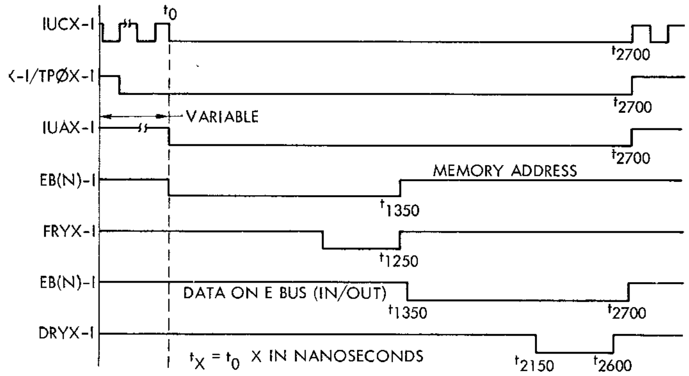
The SYRT-I signal is provided for initializing I/O controllers when the "system reset" switch is pressed on the computer console. The SYRT-I signal drops to ground when pressed, and returns to +3 volts when released. This signal is connected to a line receiver to convert to standard Micro-VersaLOGIC voltages for use in the I/O devices.

2.4.2 Interrupt Jump (IUJP-I)

The interrupt jump signal (IUJP-I) indicates that the instruction being executed due to an interrupt request (IURX-I) is a jump-and-mark instruction. The interrupt module uses this signal to inhibit further requests. The module may then be enabled under program control.

2.4.3 Interrupt Lines (IU00-I through IU15-I)

The interrupt lines in the interrupt cable are used for communication between the I/O devices and a priority interrupt module. In the absence of any interrupts, these lines may be used for user communications.



IUCX-I - REMOVED AT t_0 RETURNS AT t_{2700}

TPIX-I/TPOX-I - NORMALLY REMOVED AT t_{2700}, MUST BE REMOVED BY t_{2900}

EB(N)-I DATA (IN) - NORMALLY ON t_{1350}, MUST BE ON t1900

NORMALLY OFF t_{2600}, MUST BE OFF t2900

EB(N)-I ADDRESS - NORMALLY ON t0, MUST BE ON t600

NORMALLY OFF t1350, MUST BE OFF t1950

EB(N)-I DATA (OUT) - WILL BE ON t1500, WILL BE REMOVED t2700

NOTE SIGNAL INVERSION ON I/O LINES

Figure 2-17. Timing Sequence of Trap Input/Output.

APPENDICES

APPENDIX A
DATA 620/i NUMBER SYSTEM

DATA 620/i NUMBER SYSTEM

Binary numbers in the DATA 620/i are represented in 2's-complement form. Single-precision numbers are 15 bits plus sign (16-bit configuration) or 17 bits plus sign (18-bit configuration). The sign bit occupies the most-significant bit position (15 or 17). A "0" in the sign position denotes a positive number; a "1" in the sign position denotes a negative number. The negative of a positive number is represented in 2's-complement form.

The 2's-complement of a number may be found in either of two ways:

a. Take the 1's-complement of the number (i.e., complement each bit); add "1" in the least-significant bit position. Example:

+9	0000000000001001
1's-complement	1111111111110110
	+1
2's-complement (-9)	1111111111110111

b. For an n-bit number (including sign) subtract it from 2^{n+1}. Example:

2^{n+1}	10000000000000000
-(+9)	-0000000000001001
-9	1111111111110111

It is generally convenient to express binary numbers by their octal equivalent. This conversion is easily performed by grouping the binary bits by threes, starting with the least-significant bit. Thus, in the 18-bit configuration, numbers may be expressed by six full octal digits ($000000-777777_8$).

In the 16-bit configuration, the range of octal numbers is less than six full digits ($000000-177777_8$). The octal equivalents for the above examples are:

DECIMAL	OCTAL
+9	000011_8
-9	177767_8

The range of numbers in the DATA 620/i is from -2^{15} to $+2^{15}-1$ for the 16-bit configuration and -2^{17} to $+2^{17}-1$ for the 18-bit configuration. The zero minus 1 and plus/minus full-scale numbers for the 16-bit configuration are:

BINARY	OCTAL	DECIMAL	
0111111111111111	077777_8	+32,767	+Full Scale
0000000000000000	000000	0	0
1111111111111111	177777_8	-1	-1
1000000000000000	100000_8	-32,768	-Full Scale

The negative of the octal equivalent number is found by subtracting the number from 177777_8 and adding 1 in the least-significant digit (subtract from 777777_8 for the 18-bit configuration). Example:

	177777_8
-(9)	-000011_8
	+1
(-9)	177767_8

In performing addition or subtraction, it is possible for the results to exceed the ± full scale range of the machine. For example:

DECIMAL	OCTAL	
+21,980	052734_8	
+11,843	$+027103_8$	
33,823	102037_8	-31,713

The negative result is in error. The same type of error occurs if the sum of the two negative numbers exceeds the minus full-scale range:

DECIMAL	OCTAL	
-21,980	125044_8	
(+)-11,843	150675_8	
-33,823	$(1)075741_8$	31,803

Note that the carry out of the most-significant octal digit position is generally lost. However, to inform the programmer that the true result of an addition/subtraction falls outside the range of the machine, an overflow indicator is provided. The overflow indicator is set if the sign bit changes when two numbers of the same sign are added together (where the sign of the subtrahend is changed in subtraction).

In multiplication, a double-length product is formed in the arithmetic registers (A or B). Since the product cannot exceed 32-bits (36-bits in the 18-bit configuration), overflow will never occur as the result of a multiply. The sign of the product is automatically determined.

Example:

DECIMAL	OCTAL
21,980	052734
X 11,843	027103
65,940	200624
87,920	52734
175,840	454404
21,980	125670
21,980	
260,299,140	001741000224
	A B

The double-length result is accumulated in the A and B registers.

In division, an overflow (underflow) can occur if the divisor is less than or equal to the dividend.

APPENDIX B
STANDARD DATA 620/i SUBROUTINES

APPENDIX B
STANDARD DATA 620/i SUBROUTINES

SUBROUTINES	LOCATIONS	TIME
Elementary Functions*		
Log^e (1 + X), (0 ≤ X < 1)	19	365 usec
Exponential (e^{-X}) (0 ≤ X < 1)	17	283 usec
Exponential (e^{+X}) (0 ≤ X < 1)	17	333 usec
Square Root (0 ≤ X < 1)	58	493 usec
Sine X (-π < X < π)	31	315 usec
Cosine X (-π < X < π)	20	310 usec
Arctan (-1 to 1)	15	380 usec
Single Precision (fixed point)		
Multiply (optional)	hardware	18 usec
Divide (optional)	hardware	27 usec
Divide (programmed)	27	300 usec
Double Precision (fixed point)		
Open		
Addition	7	20 usec
Subtraction	7	20 usec
Multiplication	16	97.2 usec
Divide	28	1036 usec
Closed		
Addition	23	54.0 usec
Subtraction	25	57.6 usec
Multiply	36	127.8 usec
Divide	35	1050 usec

*All elementary functions exept square root require a subroutine called POLY, which takes 42 locations.

SUBROUTINES	LOCATIONS	TIME
Conversion		
Binary-to-BCD (4 characters)	32	249 usec
BCD-to-Binary	28	205 usec

APPENDIX C
TABLE OF POWERS OF TWO

Table of Powers of Two

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

APPENDIX D
OCTAL-DECIMAL INTEGER CONVERSION TABLE

OCTAL-DECIMAL INTEGER CONVERSION TABLE

0000 to 0777 (Octal) | 0000 to 0511 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777 (Octal) | 0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

OCTAL-DECIMAL INTEGER CONVERSION TABLE

2000 to 2777 (Octal) | 1024 to 1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 to 3777 (Octal) | 1536 to 2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

4000 to 4777 (Octal) | 2048 to 2559 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 to 5777 (Octal) | 2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

6000 to 6777 (Octal) | 3072 to 3583 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000 to 7777 (Octal) | 3584 to 4095 (Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX E
OCTAL-DECIMAL FRACTION CONVERSION TABLE

Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

APPENDIX F
DATA 260/i INSTRUCTIONS (ALPHABETICAL ORDER)

APPENDIX F
DATA 620/i INSTRUCTIONS (ALPHABETICAL ORDER)

MNEMONIC	OCTAL	DESCRIPTION	WDS/ INST	TIME CYCLES	INDIRECT ADDRESS
ADD	120000	Add to A Register	1	2	Yes
ADDE*	006120	Add to A Register Extended	2	3	Yes
ADDI	006120	Add to A Register Immediate	2	2	No
ANA	150000	AND to A Register	1	2	Yes
ANAE*	006150	AND to A Register Extended	2	3	Yes
ANAI	006150	AND to A Register Immediate	2	2	No
AØFA	005511	Add OF to A Register	1	1	No
AØFB	005522	Add OF to B Register	1	1	No
AØFX	005544	Add OF to X Register	1	1	No
ASLA	004200+n	Arithmetic Shift Left A n Places	1	1+0.25n	No
ASLB	004000+n	Arithmetic Shift Left B n Places	1	1+0.25n	No
ASRA	004300+n	Arithmetic Shift Right A n Places	1	1+0.25n	No
ASRB	004100+n	Arithmetic Shift Right B n Places	1	1+0.25n	No
CIA	102500	Clear and Input to A Register	1	2	No
CIB	102600	Clear and Input to B Register	1	2	No
CPA	005211	Complement A Register	1	1	No
CPB	005222	Complement B Register	1	1	No
CPX	005244	Complement X Register	1	1	No
DAR	005311	Decrement A Register	1	1	No
DBR	005322	Decrement B Register	1	1	No

*Optional Instructions

MNEMONIC	OCTAL	DESCRIPTION	WDS/ INST	TIME CYCLES	INDIRECT ADDRESS
DIV*	170000	Divide AB Register 16-Bit 18-Bit	1 1	10-14 11-16	Yes
DIV*	006170	Divide AB Register Extended 16-Bit 18-Bit	2	11-15 12-17	Yes
DIVI*	006170	Divide AB Register Immediate 16-Bit 18-Bit	2	10-14 11-16	No
DXR	005344	Decrement X Register	1	1	No
ERA	130000	Exclusive OR to A Register	1	2	Yes
ERAE*	006130	Exclusive OR to A Register Extended	2	3	Yes
ERAI	006130	Exclusive OR to A Register Immediate	2	2	No
EXC	100000	External Control Function	1	1	No
HLT	000000	Halt	1	1	No
IAR	005111	Increment A Register	1	1	No
IBR	005122	Increment B Register	1	1	No
IME	102000	Input to Memory	2	3	No
INA	102100	Input to A Register	1	2	No
INB	102200	Input to B Register	1	2	No
INR	040000	Increment and Replace	1	3	Yes
INRE*	006040	Increment and Replace Extended	2	4	Yes
INRI	006040	Increment and Replace Immediate	2	3	No
IXR	005144	Increment X Register	1	1	No
JAN	001004	Jump if A Register Negative	2	2	Yes
JANM	002004	Jump and Mark if A Register Negative	2	2-3	Yes

*Optional Instructions

MNEMONIC	OCTAL	DESCRIPTION	WDS/ INST	TIME CYCLES	INDIRECT ADDRESS
JAP	001002	Jump if A Register Positive	2	2	Yes
JAPM	002002	Jump and Mark if A Register Positive	2	2-3	Yes
JAZ	001010	Jump if A Register Zero	2	2	Yes
JAZM	002010	Jump and Mark if A Register	2	2-3	Yes
JBZ	001020	Jump if B Register Zero	2	2	Yes
JBZM	002020	Jump and Mark if B Register Zero	2	2-3	Yes
JMP	001000	Jump Unconditionally	2	2	Yes
JMPM	002000	Jump and Mark if Unconditionally	2	3	Yes
JØF	001001	Jump if Overflow On	2	2	Yes
JØFM	002001	Jump and Mark if Overflow On	2	2-3	Yes
JS1M	002100	Jump and Mark if Sense Switch 1 On	2	2-3	Yes
JS2M	002200	Jump and Mark if Sense Switch 2 On	2	2-3	Yes
JS3M	002400	Jump and Mark if Sense Switch 3 On	2	2-3	Yes
JSS1	001100	Jump if Sense Switch 1 On	2	2	Yes
JSS2	001200	Jump if Sense Switch 2 On	2	2	Yes
JSS3	001400	Jump if Sense Switch 3 On	2	2	Yes
JXZ	001040	Jump X Register Zero	2	2	Yes
JXZM	002040	Jump and Mark X Register Zero	2	203	Yes
LASL	004400+n	Long Arithmetic Shift Left n Places	1	1+0.50n	No
LASR	004500+n	Long Arithmetic Shift Right n Places	1	1+0.50n	No

MNEMONIC	OCTAL	DESCRIPTION	WDS/ INST	TIME CYCLES	INDIRECT ADDRESS
LDA	010000	Load A Register	1	2	Yes
LDAE*	006010	Load A Register Extended	2	3	Yes
LDAI	006010	Load A Register Immediate	2	2	No
LDB	020000	Load B Register	1	2	Yes
LDBE*	006020	Load B Register Extended	2	3	Yes
LDBI	006020	Load B Register Immediate	2	2	No
LDX	030000	Load X Register	1	2	Yes
LDXE*	006030	Load X Register Extended	2	3	Yes
LDXI	006030	Load X Register Immediate	2	2	No
LLRL	004440+n	Long Logical Rotate Left n Places	1	1+0.50n	No
LLSR	004540+n	Long Logical Shift Right n Places	1	1+0.50n	No
LRLA	004240+n	Logical Rotate Left A n Places	1	1+0.25n	No
LRLB	004040+n	Logical Rotate Left B n Places	1	1+0.25n	No
LSRA	004340+n	Logical Shift Right A n Places	1	1+0.25n	No
LSRB	004140+n	Logical Shift Right B n Places	1	1+0.25n	No
MUL*	160000	Multiply B Register 16-Bit 18-Bit	1	10	Yes
MULE*	006160	Multiply B Register Extended 16-Bit 18-Bit	2	11 15	Yes
MULI*	006160	Multiply B Register Immediate 16-Bit 18-Bit	2	10 14	No
NØP	00500	No Operation	1	1	No
ØAR	103100	Output from A Register	1	2	No
ØBR	103200	Output from B Register	1	2	No

*Optional Instructions

MNEMONIC	OCTAL	DESCRIPTION	WDS/ INST	TIME CYCLES	INDIRECT ADDRESS
ØME	103000	Output from Memory	2	3	No
ØRA	110000	Inclusive OR to A Register	1	2	Yes
ØRAE*	006110	Inclusive OR to A Register Extended	2	3	Yes
ØRAI	006110	Inclusive OR to A Register Immediate	2	2	No
RØF	007400	Reset Overflow	1	1	No
SEN	101000	Sense Input/Output Lines	2	2.25	No
SØF	007401	Set Overflow	1	1	No
SØFA	005711	Subtract OFLO from A Register	1	1	No
SØFB	005722	Subtract OFLO from B Register	1	1	No
SØFX	005744	Subtract OFLO from X Register	1	1	No
STA	050000	Store A Register	1	2	Yes
STAE*	006050	Store A Register Extended	2	3	Yes
STAI	006050	Store A Register Immediate	2	2	No
STB	060000	Store B Register	1	2	Yes
STBE*	006060	Store B Register Extended	2	3	Yes
STBI	006060	Store B Register Immediate	2	2	No
STX	070000	Store X Register	1	2	Yes
STXE*	006070	Store X Register Extended	2	3	Yes
STXI	006070	Store X Register Immediate	2	2	No
SUB	140000	Subtract from A Register	1	2	Yes
SUBE*	006140	Subtract from A Register Extended	2	3	Yes

*Optional Instructions

MNEMONIC	OCTAL	DESCRIPTION	WDS/ INST	TIME CYCLES	INDIRECT ADDRESS
SUBI	006140	Subtract from A Register Immediate	2	2	No
TAB	005012	Transfer A to B Register	1	1	No
TAX	005014	Transfer A to X Register	1	1	No
TBA	005021	Transfer B to A Register	1	1	No
TBX	005024	Transfer B to X Register	1	1	No
TXA	005041	Transfer X to A Register	1	1	No
TXB	005042	Transfer X to B Register	1	1	No
TZA	005001	Transfer Zero to A Register	1	1	No
TZB	005002	Transfer Zero to B Register	1	1	No
TZX	005004	Transfer Zero to X Register	1	1	No
XAN	003004	Execute A Register Negative	2	2	Yes
XAP	003002	Execute A Register Positive	2	2	Yes
XAZ	003010	Execute A Register Zero	2	2	Yes
XBZ	003020	Execute B Register Zero	2	2	Yes
XEC	003000	Execute Unconditionally	2	2	Yes
XØF	003001	Execute Overflow Set	2	2	Yes
XS1	003100	Execute Sense Switch 1 Set	2	2	Yes
XS2	003200	Execute Sense Switch 2 Set	2	2	Yes
XS3	003400	Execute Sense Switch 3 Set	2	2	Yes
XXZ	003040	Execute X Register Zero	2	2	Yes

*Optional Instructions

APPENDIX G
DATA 620/i INSTRUCTIONS (BY TYPE)

Table G-1
SINGLE-WORD ADDRESSED INSTRUCTIONS

Table G-1(a)
LOAD/STORE INSTRUCTION GROUP

OP CODE			
OCTAL	MNEMONIC	INSTRUCTION	TIMING (CYCLES)
01	LDA	Load A Register	2
02	LDB	Load B Register	2
03	LDX	Load X Register	2
05	STA	Store A Register	2
06	STB	Store B Register	2
07	STX	Store X Register	2

Table G-1(b)
ARITHMETIC INSTRUCTION GROUP

OP CODE			
OCTAL	MNEMONIC	INSTRUCTION	TIMING (CYCLES)
04	INR	Increment and Replace	3
12	ADD	Add Memory to A	2
14	SUB	Subtract Memory from A	2
16	MUL(*)	Multiply 16-bit	10
		18-bit	11
17	DIV(*)	Divide 16-bit	10-14
		18-bit	11-15

*Optional Instructions

Table G-1(c)
LOGICAL INSTRUCTION GROUP

OP CODE			
OCTAL	MNEMONIC	INSTRUCTION	TIMING (CYCLES)
11	ØRA	Inclusive OR, Memory and A	2
13	ERA	Exclusive OR, Memory and A	2
15	ANA	AND Memory and A	2

Table G-1(d)
ADDRESSING MODES FOR SINGLE WORD ADDRESSED INSTRUCTIONS

m FIELD				
11	10	9	ADDRESSING MODE	OPERATION
0	X	X	Direct	Combine bits 9, 10 with a field (0-8) to form effective address (0000 - 2047).
1	0	0	Relative	Add a field (bits 0-8) to contents of P to form effective address (Mod 2^{15}).
1	0	1	Index (X Register)	Add a field (bits 0-8) to contents of X to form effective address (Mod 2^{15}).
1	1	0	Index (B Register)	Add a field (bits 0-8) to contents of B to form effective address (Mod 2^{15}).
1	1	1	Indirect	a field (bits 0-8) specifies location of an address word.

Table G-2
CONTROL INSTRUCTION GROUP CODES
(SINGLE-WORD, NON-ADDRESSABLE)

OP CODE		m FIELD	a FIELD	INSTRUCTION	TIMING (CYCLES)
OCTAL	MNEMONIC				
00	HLT	0	XXX	Halt	1
00	NØP	5	000	No Operation	1
00	RØF	7	400	Reset Overflow	1
00	SØF	7	401	Set Overflow	1

Table G-3
SHIFT INSTRUCTION GROUP

Table G-3(a)
INSTRUCTION FORMAT

OCTAL	OCTAL	a FIELD								
OP CODE	m FIELD	U_8	U_7	U_6	U_5	U_4	U_3	U_2	U_1	U_0
		0 = A or B 1 = A & B	0 = B 1 = A	0 = Left 1 = Right	0 = Arith. 1 = Logical rotate	Shift Count (0 - 31)				

Table G-3(b)
INSTRUCTION FORMAT

U_8	U_7	U_6	U_5	MNEMONIC	SHIFT INSTRUCTION	TIMING (CYCLES)
0	0	0	0	ASLB	Arithmetic Shift B Left	1 + 0.25n
0	0	0	1	LRLB	Logical Rotate B Left	1 + 0.25n
0	0	1	0	ASRB	Arithmetic Shift B Right	1 + 0.25n
0	0	1	1	LSRB	Logical Shift B Right	1 + 0.25n
0	1	0	0	ASLA	Arithmetic Shift A Left	1 + 0.25n
0	1	0	1	LRLA	Logical Rotate A Left	1 + 0.25n
0	1	1	0	ASRA	Arithmetic Shift A Right	1 + 0.25n
0	1	1	1	LSRA	Logical Shift A Right	1 + 0.25n
1	0	0	0	LASL	Long Arithmetic Shift A, B Left	1 + 0.50n
1	0	0	1	LLRL	Long Logical Rotate A, B Registers Left	1 + 0.50n
1	0	1	0	LASR	Long Arithmetic Shift A, B Right	1 + 0.50n
1	0	1	1	LLSR	Long Logical Shift, A, B Registers	1 + 0.50n
1	1	0	0		Invalid	
1	1	0	1		Invalid	
1	1	1	0		Invalid	
1	1	1	1		Invalid	

Table G-4
REGISTER CHANGE INSTRUCTION GROUP

Table G-4(a)
INSTRUCTION FORMAT

OCTAL		a FIELD									TYPE OF TRANSFER
		SOURCE						DEST.			
CLASS CODE	m FIELD	U_8	U_7	U_6	U_5	U_4	U_3	U_2	U_1	U_0	
			0	0							Transfer Unchanged
			0	1							Transfer Incremented
00	5		1	0	X	B	A	X	B	A	Transfer Complemented
			1	1							Transfer Decremented

Note: Multiple source transfer results in inclusive-OR; multiple source complemented results in complement inclusive-OR.

Table G-4(b)
REGISTER CHANGE INSTRUCTION CODES

CLASS CODE FIELD OCTAL	MNEMONIC	REGISTER CHANGE INSTRUCTION	TIMING
0 0 1	TZA	Transfer Zero to A Register	1
0 0 2	TZB	Transfer Zero to B Register	1
0 0 4	TZX	Transfer Zero to X Register	1
0 1 2	TAB	Transfer A Register to B Register	1
0 1 4	TAX	Transfer A Register to X Register	1
0 2 1	TBA	Transfer B Register to A Register	1
0 2 4	TBX	Transfer B Register to X Register	1
0 4 1	TXA	Transfer X Register to A Register	1
0 4 2	TXB	Transfer X Register to B Register	1
1 1 1	IAR	Increment A Register	1
1 2 2	IBR	Increment B Register	1
1 4 4	IXR	Increment X Register	1
3 1 1	DAR	Decrement A Register	1
3 2 2	DBR	Decrement B Register	1
3 4 4	DXR	Decrement X Register	1
5 1 1	AØFA	Add Overflow to A Register	1
5 2 2	AØFB	Add Overflow to B Register	1
5 4 4	AØFX	Add Overflow to X Register	1
7 1 1	SØFA	Subtract Overflow from A Register	1
7 2 2	SØFB	Subtract Overflow from B Register	1
7 4 4	SØFX	Subtract Overflow from X Register	1

Table G-5
JUMP INSTRUCTION GROUP

Table G-5(a)
INSTRUCTION FORMAT

OCTAL		a FIELD								
OP CODE	m FIELD	U_8	U_7	U_6	U_5	U_4	U_3	U_2	U_1	U_0
00	1	SS3 ON	SS2 ON	SS1 ON	$X = 0$	$B = 0$	$A = 0$	$A < 0$	$A \geq 0$	$OF = 1$

Note: Jump condition is logical AND of all a field bits.

Table G-5(b)
JUMP INSTRUCTION CODES

a FIELD OCTAL	MNEMONIC	JUMP INSTRUCTION	TIMING (CYCLES)
0 0 0	JMP	Jump Unconditionally	2
0 0 1	JØF	Jump If Overflow Set	2
0 0 2	JAP	Jump If A Register Positive	2
0 0 4	JAN	Jump If A Register Negative	2
0 1 0	JAZ	Jump If A Register Zero	2
0 2 0	JBZ	Jump If B Register Zero	2
0 4 0	JXZ	Jump If X Register Zero	2
1 0 0	JSS1	Jump If Sense Switch 1 Set	2
2 0 0	JSS2	Jump If Sense Switch 2 Set	2
4 0 0	JSS3	Jump If Sense Switch 3 Set	2

Table G-6
JUMP AND MARK INSTRUCTION GROUP

Table G-6(a)
INSTRUCTION FORMAT

OCTAL		a FIELD								
OP CODE	m FIELD	U_8	U_7	U_6	U_5	U_4	U_3	U_2	U_1	U_0
00	2	SS3	SS2	SS1	X = 0	B = 0	A = 0	A < 0	A ≥ 0	OF = 1

Note: Jump and Mark condition is logical-AND of all a field bits.

Table G-6(b)
JUMP AND MARK INSTRUCTION CODES

a FIELD OCTAL	MNEMONIC	JUMP AND MARK INSTRUCTIONS	TIMING (CYCLES)
000	JMPM	Jump and Mark Unconditionally	2
001	JØFM	Jump and Mark if Overflow Set	2 (3 if Jump)
002	JANM	Jump and Mark if A Register Negative	2 (3 if Jump)
003	JAPM	Jump and Mark if A Register Positive	2 (3 if Jump)
010	JAZM	Jump and Mark if A Register Zero	2 (3 if Jump)
020	JBZM	Jump and Mark if B Register Zero	2 (3 if Jump)
040	JXZM	Jump and Mark if X Register Zero	2 (3 if Jump)
100	JS1M	Jump and Mark if Sense Switch 1 On	2 (3 if Jump)
200	JS2M	Jump and Mark if Sense Switch 2 On	2 (3 if Jump)
400	JS3M	Jump and Mark if Sense Switch 3 On	2 (3 if Jump)

Table G-7
EXECUTE INSTRUCTION GROUP

Table G-7(a)
INSTRUCTION FORMAT

OCTAL		a FIELD								
OP CODE	m FIELD	U_8	U_7	U_6	U_5	U_4	U_3	U_2	U_1	U_0
0 0	3	SS3 ON	SS2 ON	SS1 ON	X = 0	B = 0	A = 0	A 0	A 0	OF = 1

Note: Execute condition is logical-AND of all a field bits. Executed instruction must be single word.

Table G-7(a)
INSTRUCTION FORMAT

a FIELD OCTAL	MNEMONIC	EXECUTE INSTRUCTION	TIMING (CYCLES)
000	XEC	Execute Unconditionally	2
001	XØF	Execute if Overflow Set	2
002	XAP	Execute if A Register Positive	2
004	XAN	Execute if A Register Negative	2
010	XAZ	Execute if A Register Zero	2
020	XBZ	Execute if B Register Zero	2
040	XXZ	Execute if X Register Zero	2
100	XS1	Execute if Sense Switch 1	2
200	XS2	Execute if Sense Switch 2	2
400	XS3	Execute if Sense Switch 3	2

Table G-10
IMMEDIATE INSTRUCTION GROUP

OP CODE		OCTAL			
OCTAL	MNEMONIC	m FIELD	a FIELD	INSTRUCTION	TIMING (CYCLES)
00	LDAI	6	010	Load A Immediate	2
00	LDBI	6	020	Load B Immediate	2
00	LDXI	6	030	Load X Immediate	2
00	INRI	6	040	Increment and Replace Immediate	2
00	STAI	6	050	Store A Immediate	2
00	STBI	6	060	Store B Immediate	2
00	STXI	6	070	Store X Immediate	2
00	ØRAI	6	110	Inclusive OR Immediate	2
00	ADDI	6	120	Add Immediate	2
00	ERAI	6	130	Exclusive OR Immediate	2
00	SUBI	6	140	Subtract Immediate	2
00	MULI*	6	160	Multiply Immediate	10
00	DIVI*	6	170	Divide Immediate	10-14
00	ANAI	6	150	AND Immediate	2

*Optional Instructions

Table G-11
INPUT/OUTPUT INSTRUCTION GROUP

OP CODE		OCTAL			
OCTAL	MNEMONIC	m FIELD	a FIELD	INSTRUCTION	TIMING (CYCLES)
10	EXC	0	XZZ	External Control	1
10	SEN	1	XZZ	Program Sense	2
10	IME	2	0ZZ	Input to Memory	3
10	INA	2	1ZZ	Input to A	2
10	INB	2	2ZZ	Input to B	2
10	CIA	2	5ZZ	Clear and Input to A	2
10	CIB	2	6ZZ	Clear and Input to B	2
10	ØME	3	0ZZ	Output from Memory	2
10	ØAR	3	1ZZ	Output from A	2
10	ØBR	3	2ZZ	Output from B	2

X - Mode or logical unit number

Z - Device number

Table G-12
EXTENDED ADDRESS INSTRUCTION GROUP (Optional)

OP CODE		OCTAL		INSTRUCTION	TIMING (CYCLES)
OCTAL	MNEMONIC	m FIELD	a FIELD		
00	LDAE	6	01X	Load A Register Extended	3
00	LDBE	6	02X	Load B Register Extended	3
00	LDXE	6	03X	Load X Register Extended	3
00	STAE	6	05X	Store A Register Extended	3
00	STBE	6	06X	Store B Register Extended	3
00	STXE	6	07X	Store X Register Extended	3
00	INRE	6	04X	Increment and Replace Extended	4
00	ADDE	6	12X	Add Memory to A Register Extended	3
00	SUBE	6	14X	Subtract Memory from A Register Extended	3
00	MULE	6	16X	Multiply 16-Bit Extended	10
				Multiply 18-Bit Extended	11
00	DIVE	6	17X	Divide 16-Bit Extended	11 - 15
				Divide 18-Bit Extended	12 - 16
00	ØRAE	6	11X	Inclusive OR Extended	3
00	ERAE	6	13X	Exclusive OR Extended	3
00	ANAE	6	15X	AND Extended	3

APPENDIX H
DATA 620/i RESERVED INSTRUCTION CODES

Table H-1
INTERRUPT MODULE RESERVED INSTRUCTION CODES

The following instruction codes are for use with the first interrupt module. Device addresses 40_8 through 47_8 are reserved for interrupt modules.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
EXC 140*	100140	Clear AC Register
EXC 240	100240	Enable Interrupt Module
EXC 440	100440	Inhibit Interrupt Module
EXC 540	100540	Initialize Interrupt Module
B. TRANSFER		
OME 40	103040	Load Mask from Memory
OAR 40	103140	Load Mask from A Register
OBR 40	103240	Load Mask from B Register
C. SENSE		
None		

*AC option only

Table H-2
BIC RESERVED INSTRUCTION CODES

The following instruction codes are for use with the first buffer interlace controller. Device addresses 20_8 through 27_8 are reserved for BIC's.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
EXC 020	100020	Activate Enable
EXC 021	100021	Initialize
B. TRANSFER		
ØAR 20	103120	Load Initial Register from A
ØBR 20	103220	Load Initial Register from B
ØME 20	103020	Load Initial Register from Memory
ØAR 21	103121	Load Final Register from A
ØBR 21	103221	Load Final Register from B
ØME 21	103021	Load Final Register from Memory
INA 20	102120	Read Initial Register into A
INB 20	102220	Read Initial Register into B
IME 20	102020	Read Initial Register into Memory
CIA 20	102520	Read Initial Register into Cleared A
CIB 20	102620	Read Initial Register into Cleared B
C. SENSE		
SEN 20	101020	Sense BIC Not Busy
SEN 21	101021	Sense Abnormal Device Stop

Table H-3
TELETYPE RESERVE INSTRUCTION CODES

The following instruction codes are for use with the first teletype used in a DATA 620/i system. Device addresses 01_8 through 07_8 are reserved for teletype controllers.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
EXC 101	100101	Connect Write Register to BIC
EXC 201	100201	Connect Read Register to BIC
EXC 401	100401	Initialize
B. TRANSFER		
IAR 101	102101	Transfer Read Register to A Register
CIA 501	102501	Transfer Read Register to Cleared A Register
IBR 201	102201	Transfer Read Register to B Register
CIB 601	102601	Transfer Read Register to Cleared B Register
IME 001	102001	Transfer Read Register to Memory
OAR 101	103101	Read Write Register from A Register
OBR 201	103201	Load Write Register from B Register
OME 001	103001	Load Write Register from Memory
C. SENSE		
SEN 101	101101	Sense Write Register Ready
SEN 201	101201	Sense Read Register Ready

Table H-4
CARD READER RESERVED INSTRUCTION CODES

The following instruction codes are for use with the 90 CPM or 1100 CPM card reader. For additional card readers, device addresses will be assigned at the time of system defintion.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
EXC 230	100230	Read One Card
*EXC 630	100630	Step Read One Character
B. TRANSFER		
INA 30	102130	Transfer to A Register
INB 30	102230	Transfer to B Register
IME 30	102030	Transfer to Memory
CIA 30	102530	Transfer to A Register Cleared
CIB 30	102630	Transfer to B Register Cleared
C. SENSE		
SEN 130	101130	Sense Character Ready
SEN 230	101230	Sense Reader Not Busy
SEN 630	101630	Sense Reader Ready

*Delete for 1100 CPM reader.

Table H-5

GATED INPUT CHANNEL RESERVED INSTRUCTION CODES

The following instruction codes are for use with the gated input channel. Device addresses for additional input channels will be assigned at the time of system definition.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
None		
B. TRANSFER		
INA 60	102160	Input from Channel to A Register
INB 60	102260	Input from Channel to B Register
IME 60	102060	Input from Channel to Memory
CIA 60	102560	Input from Channel to Cleared A Register
CIB 60	102660	Input from Channel to Cleared B Register
C. SENSE		
SEN 460	101460	Sense Transfer in Request

Table H-6

BUFFER INPUT CHANNEL RESERVED INSTRUCTION CODES

The following instruction codes are for use with the buffer input channel. Device addresses for additional input channels will be assigned at the time of system definition.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
None		
B. TRANSFER		
INA 62	102162	Input from Channel to A Register
INB 62	102262	Input from Channel to B Register
IME 62	102062	Input from Channel to Memory
CIA 62	102562	Input from Channel to Cleared A Register
CIB 62	102662	Input from Channel to Cleared B Register
C. SENSE		
SEN 462	101462	Sense Transfer in Request

Table H-7
GATED OUTPUT CHANNEL RESERVED INSTRUCTION CODES

The following instruction codes are for use with the gated output channel. Device addresses for additional output channels will be assigned at the time of system definition.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
None		
B. TRANSFER		
ØAR 60	103160	Output from A Register through Channel
ØBR 60	103260	Output from B Register through Channel
ØME 60	103060	Output from Memory through Channel
C. SENSE		
SEN 260	101260	Sense Data Request

Table H-8
BUFFER OUTPUT CHANNEL RESERVED INSTRUCTION CODE

The following codes are for use with the buffer output channel. Device addresses for additional output channels will be assigned at the time of system definition.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
None		
B. TRANSFER		
ØAR 62	103162	Output from A Register through Channel
ØBR 62	103262	Output from B Register through Channel
ØME 62	103062	Output from Memory through Channel
C. SENSE		
SEN 262	101262	Sense Data Request

Table H-9
HIGH SPEED PAPER TAPE I/O RESERVED INSTRUCTION CODES

The following instruction codes are for use with the paper tape I/O unit. For additional units, device addresses will be assigned at the time of system definition. If only a reader or a punch is attached, use only those codes which apply.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
EXC 037	100037	Connect Punch to BIC
EXC 437	100437	Stop Reader
EXC 537	100537	Start Reader
EXC 637	100637	Punch Buffer
EXC 737	100737	Read One Character
B. TRANSFER		
OAR 37	103137	Load Buffer from A Register
OBR 37	103237	Load Buffer from B Register
OME 37	103037	Load Buffer from Memory
INA 37	102137	Read Buffer into A Register
INB 37	102237	Read Buffer into B Register
IME 37	102037	Read Buffer into Memory
CIA 37	102537	Read Buffer into Cleared A Register
CIB 37	102637	Read Buffer into Cleared B Register
C. SENSE		
SEN 537	101537	Sense Buffer Ready

Table H-10
MAGNETIC TAPE UNIT RESERVED INSTRUCTION CODES

The following instruction codes are for use with the first magnetic tape unit. Device addresses 10_8 through 13_8 are reserved for other magnetic tape.

MNEMONIC	OCTAL	FUNCTION
A. EXTERNAL CONTROL		
EXC 010	100010	Read One Record Binary
EXC 110	100110	Read One Record BCD
EXC 210	100210	Write One Record Binary
EXC 310	100310	Write One Record BCD
EXC 410	100410	Write File Mark
EXC 510	100510	Forward One Record
EXC 610	100610	Backspace One Record
EXC 710	100710	Rewind
B. TRANSFER		
ØAR	103110	Load Buffer from A Register
ØBR	103210	Load Buffer from B Register
ØME	103010	Load Buffer from Memory
INA	102110	Read Buffer into A Register
INB	102210	Read Buffer into B Register
IME	102010	Read Buffer into Memory
CIA	102510	Read Buffer into Cleared A Register
CIB	102610	Read Buffer into Cleared B Register
C. SENSE		
SEN 010	101010	Sense Parity Error
SEN 110	101110	Sense Buffer Ready
SEN 210	101210	Sense MTU Ready
SEN 310	101310	Sense File Mark
SEN 410	101410	Sense High Density
SEN 510	101510	Sense End of Tape
SEN 610	101610	Sense Beginning of Tape
SEN 710	101710	Sense Rewinding

APPENDIX I
STANDARD CHARACTER CODES

APPENDIX I
DATA 620/i STANDARD BCD CODES

SYMBOL	ASCII	PRINTER	MAG TAPE	HOLLERITH	FORTRAN
@	300	00	32	0-2-8	76*
A	301	01	61	12-1	13
B	302	02	62	12-2	14
C	303	03	63	12-3	15
D	304	04	64	12-4	16
E	305	05	65	12-5	17
F	306	06	66	12-6	20
G	307	07	67	12-7	21
H	310	10	70	12-8	22
I	311	11	71	12-9	23
J	312	12	41	11-1	24
K	313	13	42	11-2	25
L	314	14	43	11-3	26
M	315	15	44	11-4	27
N	316	16	45	11-5	30
O	317	17	46	11-6	31
P	320	20	47	11-7	32
Q	321	21	50	11-8	33
R	322	22	51	11-9	34
S	323	23	22	0-2	35
T	324	24	23	0-3	36
U	325	25	24	0-4	37
V	326	26	25	0-5	40
W	327	27	26	0-6	41

DATA 620/i STANDARD BCD CODES (continued)

SYMBOL	ASCII	PRINTER	MAG TAPE	HOLLERITH	FORTRAN
X	330	30	27	0-7	42
Y	331	31	30	0-8	43
Z	332	32	31	0-9	44
[	333	33	75	12-5-8	76*
\	334	34	36	0-6-8	76*
]	335	35	55	11-5-8	76*
↑	336	36	17 (Note)	7-8	76*
←	337	37	20	2-8	76[1]
blank	240	40	20	No Punch	00
!	241	41	52	11-2-8	51
"	242	42	35	0-5-8	62
#	243	43	37	0-7-8	63
$	244	44	53	11-3-8	60
%	245	45	57	11-7-8	64
&	246	46	77	12-7-8	65
'	247	47	14	4-8	66
(	250	50	34	0-4-8	52
)	251	51	74	12-4-8	53
*	252	52	54	11-4-8	47
+	253	53	60	12	45
,	254	54	33	0-3-8	54
-	255	55	40	11	46
.	256	56	73	12-3-8	51
/	257	57	21	0-1	50

DATA 620/i STANDARD BCD CODES (continued)

SYMBOL	ASCII	PRINTER	MAG TAPE	HOLLERITH	FORTRAN
0	260	60	12	0	01
1	261	61	01	1	02
2	262	62	02	2	03
3	263	63	03	3	04
4	264	64	04	4	05
5	265	65	05	5	06
6	266	66	06	6	07
7	267	67	07	7	10
8	270	70	10	8	11
9	271	71	11	9	12
:	272	72	15	5-8	67
;	273	73	56	11-6-8	70
<	274	74	76	12-6-8	76*
=	275	75	13	3-8	55
>	276	76	16	6-8	76[2]
?	277	77	72	12-2-8	76

Note: End of File for Mag Tape
* Undefined Character
1 Form Control: Return to Col. 1
2 Tab Control: Skip to Col. 7

TELETYPE CHARACTER CODES

TELETYPE CHARACTER	DATA 620/i INTERNAL CODE	TELETYPE CHARACTER	DATA 620/i INTERNAL CODE
0	260	Y	331
1	261	Z	332
2	262	blank	240
3	263	!	241
4	264	"	242
5	265	#	243
6	266	$	244
7	267	%	245
8	270	&	246
9	271	'	247
A	301	(	250
B	302	)	251
C	303	*	252
D	304	+	253
E	305	,	254
F	306	–	255
G	307	.	256
H	310	/	257
I	311	:	272
J	312	;	273
K	313		274
L	314	=	275
M	315		276
N	316	?	277
O	317	@	300
P	320		333
Q	321		334
R	322		335
S	323		336
T	324		337
U	325	Rub Out	377
V	326	NUL	200
W	327	SOM	201
X	330	EOA	202

TELETYPE CHARACTER CODES (continued)

TELETYPE CHARACTER	DATA 620/i INTERNAL CODE	TELETYPE CHARACTER	DATA 620/i INTERNAL CODE
EOM	203	X-OFF TAPE OFF	223
EOT	204	AUX	224
WRU	205	ERROR	225
RU	206	SYNC	226
BEL	207	LEM	227
FE	210	SO	230
H TAB	211	S1	231
LINE FEED	212	S2	232
V TAB	213	S3	233
FORM	214	S4	234
RETURN	215	S5	235
SO	216	S6	236
SI	217	S7	237
DCO	220		
X-ON	221		
TAPE AUX ON	222		

APPENDIX J
TELETYPE I/O INSTRUCTIONS

APPENDIX J
TELETYPE I/O INSTRUCTIONS

I. MODEL A TELETYPE INSTRUCTIONS

A. External Control

EXC 000	100000	Select High Speed Input
EXC 100	100100	Select Paper Tape Input
EXC 200	100200	Select Keyboard Input
EXC 300	100300	Select Page and/or Paper Tape Out
EXC 400	100400	Select Off

B. Transfer

OAR 00	103100	Transfer A Register to TTY Buffer
OBR 00	103200	Transfer B Register to TTY Buffer
OME 00	103000	Transfer Memory to TTY Buffer
INA 00	102100	Transfer TTY Buffer to A Register
INB 00	102200	Transfer TTY Buffer to B Register
IME 00	102000	Transfer TTY Buffer to Memory
CIA 00	102500	Transfer TTY Buffer to A Register cleared
CIB 00	102600	Transfer TTY Buffer to B Register cleared

C. Sense

SEN 000	101000	Sense TTY Not Busy
SEN 100	101100	Sense TTY Buffer Ready
SEN 300	101300	Sense TTY Reader Ready

II. MODEL B* TELETYPE INSTRUCTIONS

A. External Control

EXC 101	100101	Connect Write Register to BIC
EXC 201	100201	Connect Read Register to BIC
EXC 401	100401	Initialize

B. Transfer

OAR 101	103101	Transfer A Register to Write Register
OBR 201	103201	Transfer B Register to Write Register
OME 001	103001	Transfer Memory Register to Write Register
IAR 101	102101	Transfer Read Register to A Register
IBR 201	102201	Transfer Read Register to B Register
IME 001	102001	Transfer Read Register to Memory Register
CIA 501	102501	Transfer Read Register to Cleared A Register
CIB 601	102601	Transfer Read Register to Cleared B Register

C. Sense

SEN 101	101101	Write Register Ready
SEN 201	101201	Read Register Ready

D. Teletype Command Codes

FUNCTION	SYMBOL	CODE	TYPED AS
Print Enable	SOM	201	Control and A
Print Suppress	EOT	204	Control and D
Reader On	XON	221	Control and Q
Punch On	TAPE	222	Control and R
Reader Off	XOFF	223	Control and S
Punch Off	TAPE OFF	224	Control and T

*The following models are B-type teletypes:

620-60B	(ASR-33 TM)
620-61B	(ASR-35 TM)
620-62B	(ASR-35 TM)

III. TELETYPE CONTROL AND TRANSMISSION CODES

FUNCTION	CONTROL CODE
NUL (bcd)	200
SØM (print on)	201
EOA	202
EOM	203
EOT (print off)	204
WRU	205
RU	206
BEL	207
FE	210
HTAB	211
LINE FEED	212
V TAB	213
FORM	214
CARRIAGE RETURN	215
SO	216
SI	217
DCO	220
X-ON (reader on)	221
TAPE (punch on)	222
X-OFF (reader off)	223
TAPE OFF (punch off)	224
ERROR	225
SYNC	226
LEM	227
S 0	230
S 1	231
S 2	232
S 3	233
S 4	234
S 5	235
S 6	236
S 7	237

APPENDIX K
FORTRAN STATEMENT TYPES

APPENDIX K
FORTRAN STATEMENT TYPES

STATEMENT	EXECUTABLE	NON-EXECUTABLE
ARITHMETIC ASSIGNMENT	X	
BACKSPACE	X	
CALL	X	
COMMON		X
CONTINUE	X	
DIMENSION		X
DO	X	
END		X
ENDFILE	X	
EQUIVALENCE		X
FORMAT		X
FUNCTION		X
GO TO	X	
IF	X	
PAUSE	X	
READ	X	
RETURN	X	
STOP	X	
SUBROUTINE		X
WRITE	X	

APPENDIX L
FORTRAN I/O UNIT ASSIGNMENTS

APPENDIX L

FORTRAN I/O UNIT ASSIGNMENTS

The following logical unit numbers are associated with the indicated devices at execution time.

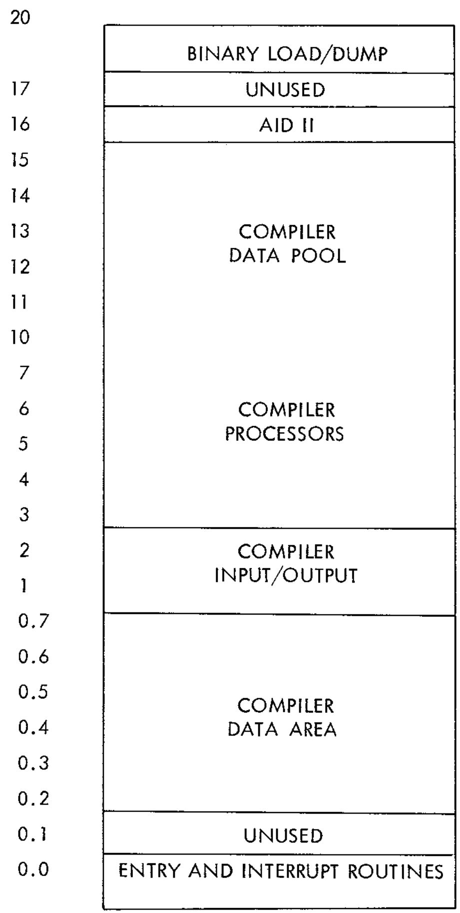
Logical Unit 0:	Teletype keyboard and printer
Logical Unit 1:	Teletype paper tape reader and punch
Logical Unit 2:	High speed paper tape reader/punch
Logical Unit 3:	Card reader/punch
Logical Unit 4:	Line printer
Logical Unit 8:	Magnetic tape unit 0
Logical Unit 9:	Magnetic tape unit 1
Logical Unit 10:	Magnetic tape unit 2
Logical Unit 11:	Magnetic tape unit 3

APPENDIX M

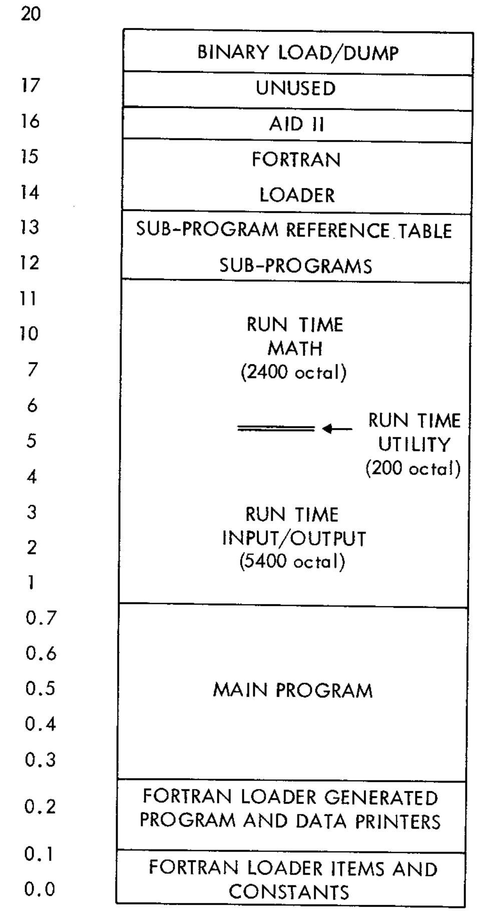
FORTRAN MEMORY MAPS

APPENDIX M
FORTRAN MEMORY MAPS

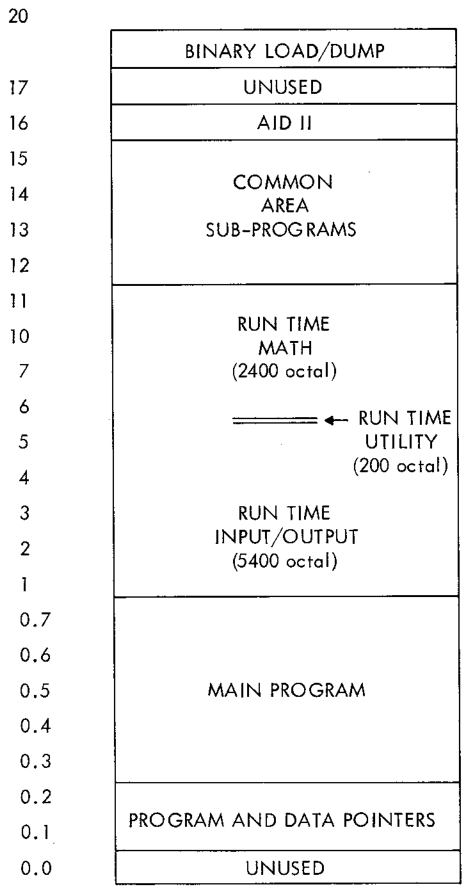
M.1 COMPILE MEMORY MAP



M.2 LOAD TIME MEMORY MAP



M.3 EXECUTION TIME MEMORY MAP



APPENDIX N

FORTRAN OBJECT RECORD FORMAT

APPENDIX N
FORTRAN OBJECT RECORD FORMAT

General

Each FORTRAN generated program will consist of a series of records, the first of which is marked as the first record of the program. All programs are terminated by an end of program word, and for main programs, an end of tape record. If a program is a function or a subroutine, the first data field of the first record will contain the subprogram name and entry address.

Record Structure

FORTRAN object records are a fixed length of 64 words. Word 1 is unused. Word 2 is the record control word. Words 3 through 5 contain the program name. Words 6 through 63 contain data fields. Word 64 contains the checksum, which is the exclusive OR of words 1 through 63.

Record Control Word Format

BIT 0:	Checksum is present
BIT 1:	End of tape
BIT 2:	End of program
BIT 3:	Start of program
BIT 4:	FORTRAN main program
BIT 5:	FORTRAN subprogram
BIT 6:	Machine language subprogram.

Program Name Format

Six 6-bit characters in packed FORTRAN format. High order starting at bit 3 of word 3, low order ending at bit 0 of word 5. (Bits 16 and 17 unused.)

Data Field Format

Data fields are either two or four word entries. Two word entries consist of a control word and a data word. Four word entries consist of a control word, two name words and a data word.

Control Word Format

CODE	SUBCODE	POINTER	NAME
15 14 13	12 11 10 9 8	7 6 5 4	3 2 1 0

Code Values

(0) Refer to subcode for specific action.

(1) Add the location of the selected pointer to the data word (2) before loading it, unless pointer 1 is specified, in which case lower the location by bits 0 through 8 of the data word.

(2) Add the value of the selected pointer to the data word (2) before loading it.

(4) Load the data word (2) absolute.

Subcode Values

(0) Ignore this entry (1 word only).

(1) Set the loading location counter to the value of the selected pointer plus the data word (2).

(2) Chain the current loading location counter value to the chain whose last location is indicated by the selected pointer and the data word (2). Stop chaining when an absolute zero address is encountered.

(6) Terminated error at compile time. Discontinue loading.

(7) Program generated successfully.

(10) Define subprogram with name and entry point given in the data word (4).

(11) Define a region for the pointer indicated whose size is given in the data word (4). Name is given for labeled common regions.

(12) Call an external subroutine with the name given. The chain address is given by the selected pointer and the data word (4).

Pointer Values

(0) Program and embedded data region.
(1) Non-common, non-embedded data region.
(2) Blank common region.
(3-31) Labeled common region (not currently implemented).

Name

The first four bits of the first character of a five-character name.

Name Format

Names are five six-bit characters starting in bit 3 of the control word and ending with bit 0 of the second name word.

Data Words

Data words contain instructions, constants, chain addresses, entry addresses, and address offset values.

Paper Tape Format

Paper tape object programs are preceded and followed by 6.4 inches of channel 8 leader. Paper tape records are preceded by a visual record mark (3 frames of rubouts, 377 octal) and a binary record mark (1 zero frame). Each word of the record is punched in three frames of paper tape, 6 bits per frame, high order first. For each frame channel 8 is not punched, channel 7 is the logical compliment of channel 6, and bits 6 through 1 are two octal digits of the word.

CHANNEL	8	7	6	5	4	3	2	1
3 FRAMES (1 word)	*	**	17	16	15	14	13	12
	*	**	11	10	9	8	7	6
	*	**	5	4	3	2	1	0

*Blank channel
**Complement of channel 6

NOTES