A Pocket Guide to the 2100 Computer

HEWLETT hp PACKARD

A Pocket Guide to the Hewlett-Packard 2100 Computer

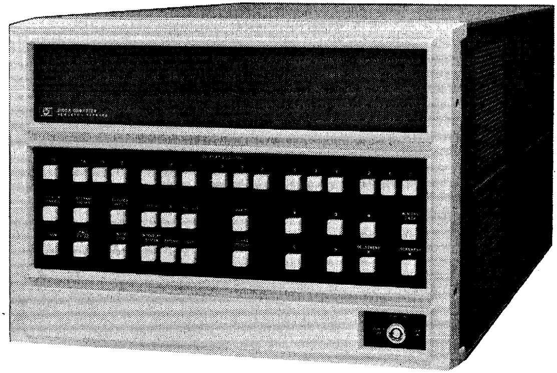
PREFACE

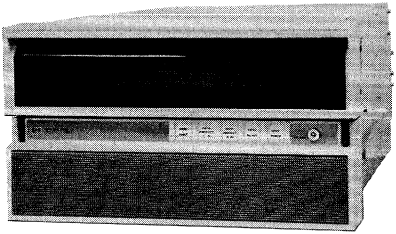
This manual combines in one convenient publication comprehensive hardware and software information for the Hewlett-Packard 2100 Computer and programmer's reference manuals for the principal software systems. Software manuals included are FORTRAN, BASIC, Assembler, and Basic Control System. System designers and programmers will find this book a handy, permanent reference. Potential users will find the technical descriptions valuable for evaluating Hewlett-Packard computers and supporting software. Since Hewlett-Packard hardware and software specifications are subject to change, the information in this manual is intended to be used strictly as a guide and does not necessarily represent current policies and products supported by Hewlett-Packard.

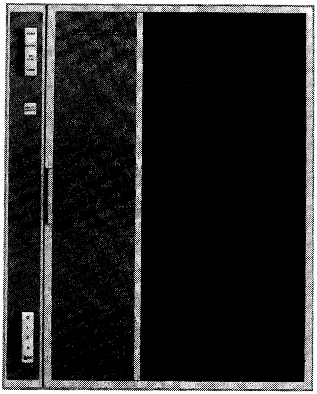
Further information on Hewlett-Packard computer products is available from your local Hewlett-Packard field office, one of more than 172 Sales and Service Offices throughout the world.

Or write Hewlett-Packard, 1501 Page Mill Road, Palo Alto, California 94304; Europe, 1217 Meyrin 2 — Geneva, Switzerland.

Here is a range of computer capabilities with the power to solve problems for a wide range of applications—at a cost that makes them uniquely practical.





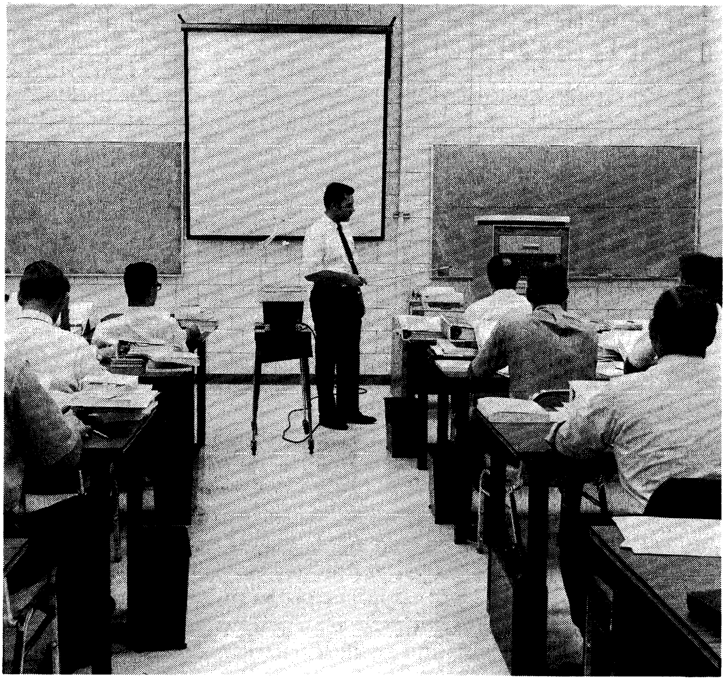


The Hewlett-Packard 2100 computer combines performance and economy with small size. Achieved by simplicity of design—in package, in hardware, in software. A package that's easy to set up, with peripherals interfaced through plug-in cards. All modular for easy expansion. Straightforward machine organization and consoles that are easy to use. The 2100 uses a microprogrammed control section that utilizes the latest in integrated circuit design. A full range of proven software packages permits your 2100 to go to work right away. All designed for busy computer users who want tomorrow's answers today.

FEATURES

- Low Cost
- Proven software
- 16-bit word size
- 980 nsec memory cycle time
- Large 1024-word page size
- Powerful instruction set of 80 basic instructions
- Peripherals interface simply with plug-in cards
- Multilevel priority interrupt for device servicing
- Two accumulators, both addressable to simplify programming
- Includes extended arithmetic instructions, power fail interrupt with auto restart and memory parity check with interrupt as standard features
- Core storage expandable to 32,768 words
- Protected loader
- Multiplexed I/O available
- Optional high-speed Direct Memory Access
- Floating point hardware option provides 5- to 20-fold performance increase of floating-point arithmetic functions
- Writeable control store available for adding additional instructions
- Modular I/O drivers—for device independent programming
- FORTRAN II and IV, Assembly, ALGOL and HP Extended BASIC
- Modular Debug package—for on-line program debugging

A LOW COST COMPUTER WITH HIGH-PRICED PERFORMANCE



USER (PROGRAMMING TRAINING)

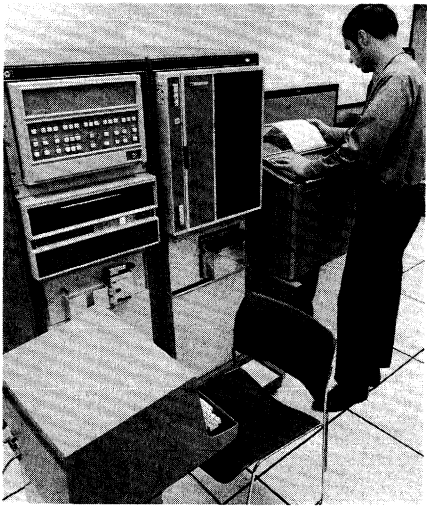
Hewlett-Packard provides a free user-programmer course for computer customers. Training materials are provided at no charge. The complete User Training Course assumes no knowledge of computer programming or electronic systems operation. It covers instruction on programming languages and operating system. At least two full days are devoted to hands-on experience.

REPAIR SERVICE

Help in maintaining your Hewlett-Packard equipment in first-rate operating condition is as close as your telephone. Service and parts assistance is available from over 140 HP field offices throughout the free world. Local service facilities are backed up by Regional Service Centers. Major parts warehouses are located in Mountain View, California, and Rockaway, New Jersey. Board exchange programs for computers and other equipment enable systems to be returned to normal operation with minimal downtime.

CONTENTS

2100A Reference Manual



The 2120 Disc Operating System lets you combine the 2100A Computer, the 7900A Disc and more than a dozen peripherals to solve your particular processing problem.

TABLE OF CONTENTS

ILLUSTRATIONS

TABLES

APPENDIX

INTRODUCTION 1

The Hewlett-Packard 2100A Computer is a compact data processor featuring a powerful, extended instruction set, plug-in interfaces, and modular software. Standard features include memory parity generation and checking, memory and I/O protect for executive systems, extended arithmetic capability, and power fail interrupt with automatic restart. Optional features include two-channel direct memory access, multiplexed input/output, a controller panel, and the I/O interfaces. The controller panel, which provides a minimum of controls and indicators, is available for applications where the full complement of controls and indicators provided on the operator panel is not necessary.

The logical design and software follow conventional standards of computer usage and notation so that the 2100A may be used as a free-standing device or in systems such as process control, media conversion, data reduction, communications, or time-sharing.

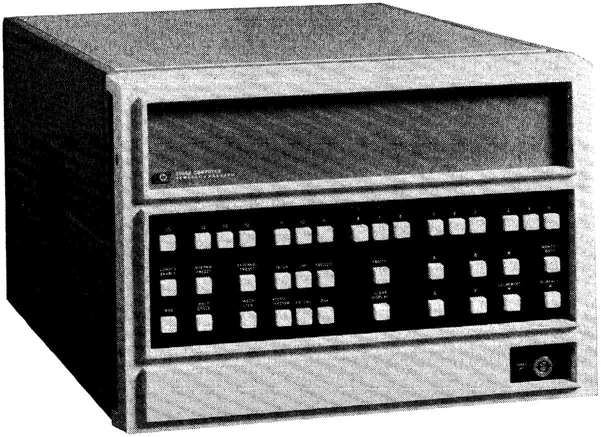


Figure 1.1. Hewlett-Packard 2100A Computer

Memory

- Magnetic core storage
- 980 nanosecond cycle time
- Parity generation and checking is standard in all units
- Six memory sizes available, 4,096 to 32,768 words; field-expandable by plug-in cards
- 1024-word page size
- Protected 64-word block for stored loader

Processor

- 80 basic instructions, including extended arithmetic
- Up to eight instructions may be combined into one word (register reference group)
- Two accumulators, addressable as memory locations
- Unlimited levels of indirect addressing allowed
- Six working registers, may be selected for display and instant modification (A, B, T, P, M, S)
- Illuminated control pushbuttons allow simultaneous display and control of internal functions
- All instructions fully executed in 1.96 microseconds, except ISZ and extended arithmetic (2.94 to 16.7 microseconds)
- Only 980 nanoseconds added for each level of indirect addressing
- Memory and I/O protection is standard

Software

- FORTRAN, FORTRAN IV, ALGOL, and BASIC languages
- Extended Assembly language
- Editor, subroutine library, Formatter, and Debug routine
- Several operating systems, including:
  Basic Control System
  Magnetic Tape System
  Disc Operating System
  Time-Shared BASIC System

Input/Output System

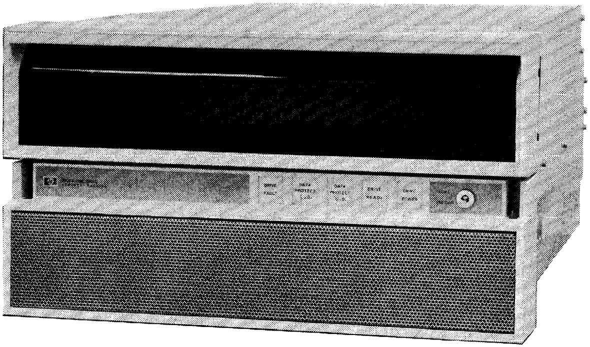
- 14 internal I/O channels, externally expandable to 45
- Optional multiplexed I/O extends capabity to 56 channels; may be plugged into any slot
- All channels buffered and bi-directional
- Multilevel priority interrupt for device servicing
- Peripherals interfaced simply with plug-in cards
- Optional dual-channel direct memory access, can transfer 1,020,400 words per second
- General-purpose interface cards available

Table 1.1. 2100A General Specifications

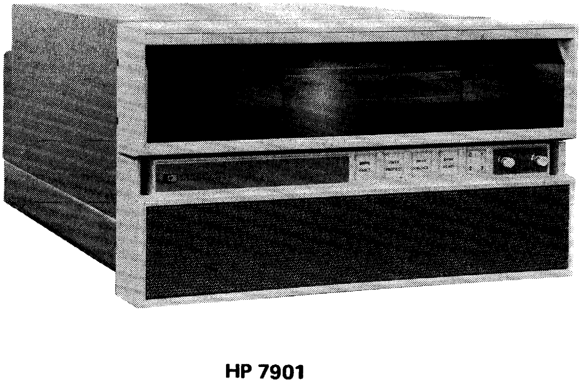
Peripherals

- Magnetic Tape
  Read and write 9-track IBM-compatible magnetic tape, 800 and 1600 cpi, at speeds of 25, 37.5, or 45 inches per second; also read and write 7-track IBM-compatible magnetic tape at speeds of 25, 37.5, or 45 inches per second with switch-selectable densities of 200, 556, and 800 cpi.
- Disc Memory
  Fixed head-per-track design for rapid access, capacities range from 262,144 to 1,048,576 words
- Cartridge Disc
  Moving-head disc for low-cost mass storage; capacities from 2.5 million to 4.9 million bytes
- Disc File
  Moving-head mass storage; 11.7 million words per drive, 8 drives maximum
- Card Reader
  Reads punched 80-column cards, 12 bits in parallel, at 1000 cards per minute
- Mark Reader
  Reads punched and pencil-marked cards at 200 cards per minute
- Line Printers
  Print 120 or 132 columns per line at 300 or 600 lines per minute; ASCII 64-character set. Also from 356 lines per minute (80 columns) to 1110 lines per minute (20 columns); 64-character set
- Keyboard Display Terminal
  CRT screen displays 25 lines, 72 characters per line; standard teleprinter keyboard plus 10-key numerical keyboard; speeds of 10 to 200 characters per second, switch selectable
- Tape Readers
  Read 5- and 8-level punched paper tape at up to 500 characters per second; with or without automatic reroller
- Tape Punch
  Punches 5- and 8-level code at 120 characters per second; also 5- and 8-level code at 75 characters per second

Table 1.2. 2100A Peripherals



HP 7900



HP 7901

Figure 1.2. The HP 7900 and 7901 Cartridge Disc Drives allow the 2100 user to economically and efficiently add on-line mass storage capability. The 7900 provides 4.9 million bytes of storage and an average access time of 30 milliseconds.

1.1 INTERFACING

Interfacing of peripheral devices is accomplished by plug-in interface cards. The computer mainframe can accommodate up to 14 interface cards, expandable to a total of 45 when the optional 2155A I/O Extender is used. Interrupt and addressing capabilities are present for 56 channels so that, using multiplexed I/O and an external controller, up to 56 devices can be handled. Interface cards are available for a wide variety of peripheral devices, and virtually all interfaces used in 2114/2115/2116-series computers may be used with the 2100A Computer. No power supply extenders are necessary for any combination of interfaces installed.

All I/O channels are buffered and bi-directional, and are serviced through a multilevel priority interrupt structure. The two optional

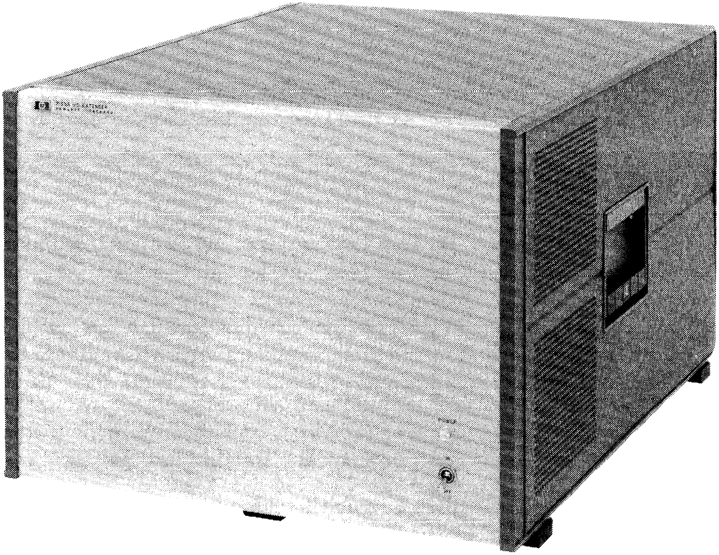


Figure 1.3. The HP 2155A Input/Output Extender adds 31 additional I/O slots to the 2100. Full interrupt and addressing capabilities are included, plus sufficient power for any combination of interfaces.

direct memory access (DMA) channels are program-assignable to any two of the 14 interface slots in the mainframe, expandable to 45 slots if DMA is also installed in the extender, and can be dynamically reassigned. DMA transfers occur on a cycle-stealing basis, not subject to the I/O priority structure. The total bandwidth through both DMA channels is more than one million words per second.

A unique channel identification and service priority interrupt is provided for every input/output channel used. Priority levels of the peripheral equipment connected to the computer can be altered simply by changing the positions of the interface cards in the I/O slots. Virtually every Hewlett-Packard measurement instrument provides a digital data output that can be interfaced to the 2100.

a. Digital voltmeters and associated signal converters for measuring dc and ac voltages, currents, and resistances. With suitable transducers, physical quantities such as pressures, loads, temperatures, and fluid flows can be measured with an HP computer.

b. Electronic counters for frequency or period measurements from a few cycles per second into the microwave region.

c. Scaler timers for nuclear radiation measurements.

d. Digital test subsystems for measurement of integrated circuits, p.c. cards, components or assembled equipment.

Analog input scanners are available for multiplexing signals into these measuring instruments. Digital scanners are also available for applications where it is desirable to multiplex the data outputs of these instruments before entry into the computer. Complete information on HP computer peripherals and measurement instrumentation are available from your local HP field sales office.

Off-the-shelf interface cards enable the customer to operate a wide variety of devices of his own choosing with the 2100. These include 8- or 16-Bit Duplex Register cards, Microcircuit Interface card, a Relay Output card, a D-to-A Converter card, and Multiplexed Input/Output for connection of up to 56 devices to the 2100.

1.2 INPUT/OUTPUT DEVICES

Instructions or data may be entered on punched tape through a teleprinter, keyboard-display terminal, high-speed photoelectric tape reader or card reader. Data output devices include the teleprinter, which provides typewritten and punched tape records, tape punches, magnetic tape units (for IBM-compatible, 7- and 9-channel recording) and line printers. Fixed-head disc or removable disc storage units are available for on-line mass storage requirements. Data can be entered on-line from Hewlett-Packard data sources and computed in real-time, or recorded on punched tape, magnetic tape, or disc for subsequent computer processing. Data-Set interfaces are also available, which enable information to be transmitted over the telephone system, into or out of the HP computer.

1.3 SOFTWARE

Software for the 2100 Computer includes four high-level programming languages: HP FORTRAN, HP FORTRAN IV, HP ALGOL, and HP BASIC, plus an efficient, extended assembler which is callable by FORTRAN and ALGOL. Utility software includes a debugging routine, a symbolic editor, and a library of commonly used computational procedures such as Boolean, trigonometric, and plotting functions, real/integer conversions, natural log, square root, etc.

Hewlett-Packard provides several systems built around BASIC interpreters. The single-terminal BASIC system allows the user to prepare and run BASIC language programs conversationally through a teleprinter. Programs can also be entered through a tape reader and punched out on tape punches. A memory of at least 8K words is required. A similar system, Educational BASIC, allows BASIC programs to be translated from marked cards.

Several operating systems are available, covering a wide range of applications. The Basic Control System, which simplifies the control of input/output operations, also provides relocatable loading and linking of user programs. The time-shared systems, using conversational BASIC language, permit 16 or 32 terminals to be connected to the system, either directly or by telephone lines via Dataphones. The Hewlett-Packard Real-Time Executive (RTE) system permits several programs to run in real-time concurrently with general-purpose background programs. This allows multiple data processing capabilities where separate computers are not economically feasible. The user can write programs in HP Assembly. FORTRAN, or ALGOL languages. A Magnetic Tape System and a Disc Operating System are also available. These systems greatly increase the speed and simplicity of assembling, compiling, loading, and executing user programs.

1.4 SYSTEM EXPANSION FEATURES

Memory sizes for the 2100A Computer are available in six configurations: 4K, 8K, 12K, 16K, 24K and 32K. All core memory is accommodated in the computer main-frame and is field-installable.

Figure 1.4 illustrates the configuration of the basic 2100A Computer and the expansion capabilities of memory and input/output. This figure approximately represents the top view and layout of the computer. For 4K or 8K memory, a card with the appropriate stack configuration is installed in position A. For 12K or 16K

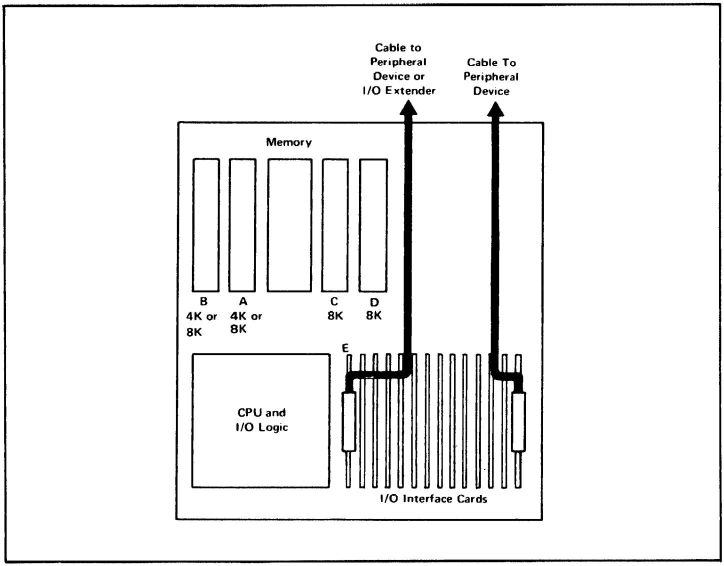


Figure 1.4. Internal Configuration

memory, the appropriate combination of 4K and 8K stacks is installed in positions A and B. For 24K, positions A and B have 8K stacks and an 8K stack is added in position C. For 32K, a final 8K stack is added in position D.

Expansion of input/output beyond the capability of the mainframe is accomplished by plugging an extender interface card into the highest address I/O slot (represented by E in figure 1.4), in place of an I/O interface card. This card is then cabled to an equivalent card in the 2155A I/O Extender Unit. The address formerly assigned to slot E, and all higher addresses, are available in the extender.

1.5 FLOATING POINT HARDWARE

The Floating Point Hardware option (12901A) supplies six additional arithmetic instructions in the 2100's basic instruction set. These instructions provide a 5- to 20-fold increase in the performance of floating point arithmetic functions. Firmware coding is stored in bipolar Read-Only-Memories (ROM's) contained in the microprocessor of the 2100.

Floating Point Hardware may be used with the 2100 Basic Control System, Magnetic Tape System, Disc Operating System or Real-Time Executive System. It can be either field or factory installed and includes an Assembler, Cross Reference Symbol Table Generator, Program Library, and a Diagnostic for the appropriate operating system and memory size.

1.6 MICROPROGRAMMING THE 2100A

Microprogramming allows the 2100's basic instruction set to be tailored to specific applications. Control storage in the 2100 consists of 1024 24-bit words organized into four modules. Microprograms for the basic 2100 instruction set are contained in the first 256-word module. A total of 768 words is available for extensions to the basic instruction set. (Firmware for the 12901A Floating Point option is stored in the first module and is reserved for this purpose.)

The Writeable Control Store (WCS) option (12908A) provides the capability to microprogram the 2100 easily and conveniently. WCS consists of a single card which plugs into a computer I/O slot eliminating extensive cabling or additional power supply requirements. The card contains 256 24-bit words of Random-Access-Memory, including all necessary address and read/write circuits.

WCS can be programmed and verified under computer control using standard input/output instructions. WCS is read at full speed via a flat cable connecting it to the control section of the computer. Up to three WCS cards may be included for development and execution of user microcode. Software supplied with WCS includes a micro-assembler, utility and I/O routines, drivers and diagnostics. The microassembler and utility routines require 8K of core (12K for use with a disc-operating system). Once developed, microprograms will operate in any core size.

The 12909A PROM Writer allows a user to convert microprograms developed with WCS to Read-Only-Memory, which can then be added to the control section of the computer. Programmable ROM's provide an economical way to reproduce debugged instruction extensions once dynamic WCS is no longer required.

The PROM Writer is located on a single card which fits in a computer I/O slot. This allows the PROM Writer to be implemented without extensive cabling or additional power supply. A stand-alone computer program, supplied with the PROM Writer, writes and verifies PROM chips using a punched tape. An 8K memory is required.

1.7 PHYSICAL SPECIFICATIONS

1.7.1 POWER REQUIREMENTS

a. Line Voltage: 115 Vac (±10%), single phase 12A or
230 Vac (±10%), single phase 6A

b. Line Frequency: 47.5 to 66 Hz

c. Computer power consumption with internal supplies loaded to capacity by plug-in options: 800 watts

d. Power Cable: 10 feet, NEMA Type 5-15P (115 Vac operation) or NEMA Type 6-15P (230 Vac operation)

1.7.2 CURRENT AVAILABLE TO I/O

Voltage	2100A Mainframe	2155A Mainframe
+4.85 V	16.8 A	45.8 A
- 2 V	7.0 A	19.5 A
+12 V	3.0 A	5.0 A
-12 V	3.0 A	5.0 A

1.7.3 ENVIRONMENTAL LIMITS*

a. Operating Temperature: 0° to 55°C (+32° to +131F)

b. Relative Humidity: 50 to 95% at 25° to 40°C (+77° to +104°F) without condensation

1.7.4 VENTILATION

a. Intake: Rear panel

b. Exhaust: Sides of front panel and cabinet

c. Air Flow: 400 cubic feet per minute

d. Heat Dissipation: 2300 BTU/hour maximum

1.7.5 ALTITUDE*

a. Operating: 15,000 feet

b. Non-operating: 25,000 feet

1.7.6 DIMENSIONS*

a. Width: 16¾ inches (42,5 cm) with adapters for mounting in 19 inch (48.3 cm) rack

b. Height: 12¼ inches (31,1 cm) (rack mounted)

c. Depth:

2100A 26 inches (66 cm), 23 inches (58,4 cm) behind rack mounting ears
2155A 23½ inches (59,6 cm), 23 inches (58,4 cm) behind rack mounting ears

*Except as noted, specifications apply to both the 2100A and the 2155A I/O Extender.

1.7.7 CLEARANCE REQUIRMENTS

a. Recommended Cable Clearance at Rear: 5 inches (127 mm) minimum

b. Recommended Air Exhaust at Top: 3 inches (76,2 mm) minimum

c. Recommended Air Exhaust at Sides: 2 inches (50,8 mm) minimum

1.7.8 WEIGHT

a. Minimum: 92 pounds (41 Kg)

b. Maximum: 115 pounds (52,2 Kg) with 32K and all I/O slots filled

1.7.9 SERVICE ACCESS

a. Top panel slides back and up permitting top access to input/output connectors, test switches, plug-in circuit boards, and wiring.

b. Bottom panel is removable for access to backplane wiring.

1.8 SYSTEM DOCUMENTATION

Full hardware documentation is provided with each computer shipped to a customer and consists of five volumes as follows:

a. 2100A Reference Manual. This manual describes the specifications, operating instructions and programming information for the computer. (The first section of this pocket manual includes the information supplied in the reference manual.)

b. Installation and Maintenance Manual. The I and MM contains instructions for installation, maintenance, troubleshooting and repair, except as covered in the power supply manual.

c. Diagrams Manual. This manual provides interconnecting information and schematic diagrams for all assemblies of the computer except the power supply.

d. IPB Manual. Replaceable parts ordering information, replaceable parts lists, exploded views, part location diagrams, and numerical lists of parts for all assemblies of the computer except the power supply are covered in the IPB manual.

e. Power Supply Manual. The power supply manual contains information necessary to troubleshoot and repair the power supply. This includes installation instructions, schematic diagrams, and replaceable parts information.

Information on microprogramming the 2100 is contained in two publications. A 2100 Microprogramming Guide (5951-3028) serves as a complete reference on how to use the microprogramming capability of the 2100. Microassembler documentation is also required in order to format and assemble microprograms correctly. A software microprogramming guide (02100-90133) describes the various aspects of microprogramming software.

All software supplied with HP computer systems is supported by complete user documentation. General types of software manuals include language manuals, operating system manuals, software operating procedures, user manuals, applications manuals, and small program manuals. A "Software Installation Record" supplied with each system lists all software furnished with the original equipment and provides an index to the software documentation. Software manuals typically sent with a 2100 computer system are listed below. (This pocket manual includes the first four of the listed reference manuals.)

1. HP Assembler
2. Basic Control System
3. HP FORTRAN
4. HP BASIC
5. ALGOL
6. Operating System Manual (Disc Operating System, Real-Time Executive System, or Magnetic Tape System, etc.)
7. Symbolic Editor
8. Relocatable Subroutines
9. System Operating Procedures

In addition to the manuals shipped with each computer, a manual titled "Preface to Programming" (5951-1354) is also available. This manual is designed to provide a general introduction to the types of languages, operating systems, and user aids available for the 2100 computer line.

PROGRAMMING INFORMATION 2

2.1 DATA FORMATS

The basic data format for the 2100 Computer is a 16-bit word. Bit positions are numbered from 0 through 15, in order of increasing significance. Bit position 15 of the data format is used for the sign bit; a "0" in this position indicates a positive number and a "1" indicates a negative number. The data is assumed to be a whole number, thus the binary point is assumed to be to the right of the number.

The basic word, shown in figure 2.1, can also be divided into two 8-bit bytes or combined to form a 32-bit doubleword. The byte format is used for character-oriented input/output devices. Packing of the two bytes into one word is accomplished by the software drivers. In I/O operations the higher order byte (Byte 1) is the first to be transferred.

The integer doubleword format is used for extended precision arithmetic in conjunction with the ten extended arithmetic instructions. Bit 15 of the most significant word is the sign bit, and the binary point is assumed to be to the right of the least significant word.

The floating point doubleword format is used with floating point software. Bit 15 of the most significant word is the mantissa sign bit and bit 0 of the least significant word is the exponent sign bit. Bits 1 through 7 are used to express the exponent, and the remaining bits (8 through 15 of the least significant word and 0 through 14 of the most significant word) are used to express the mantissa. The mantissa is assumed to be a fractional value, thus the binary point appears to the left of the mantissa. Software converts decimal numbers to this binary form and normalizes the quantity expressed (sign and leading mantissa bit differ). If either the mantissa or the exponent is negative, that part is stored in two's complement form. The number must be in the approximate range of 10^{-38} through 10^{+38}.

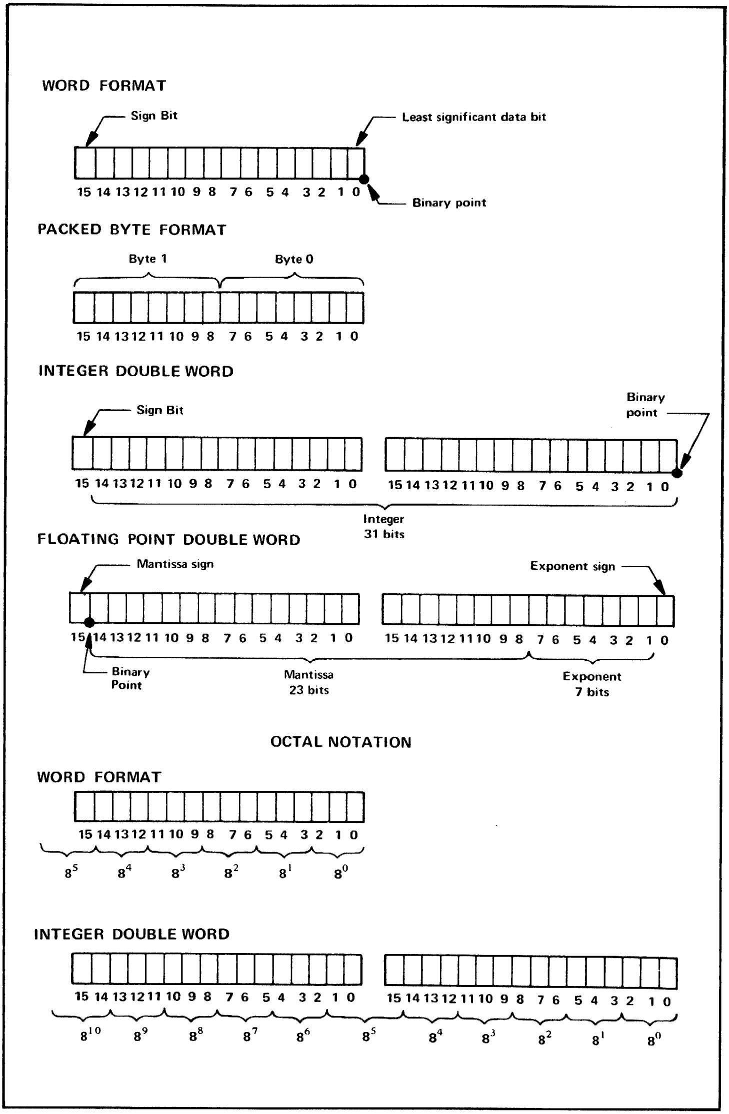


Figure 2.1. Data Formats and Octal Notation

Figure 2.1 also illustrates the octal notation of data for both single-length and double-length words. Each group of three bits, beginning at the right, is combined to form an octal digit. Each digit to the left increases in significance. A single-length 16-bit word can therefore be fully expressed by six octal digits and a double-length 32-bit word can be fully expressed by 11 octal digits. Octal notation is not shown for byte or floating point formats, since bytes normally represent characters and floating point numbers are given in decimal form.

For single-word data, the range of representable numbers is +32,767 to -32,768 (decimal), or +77,777 to -100,000 (octal). For doubleword integer data, the range is +2,147,483,647 to -2,147,483,648 (decimal), or +17,777,777,777 to -20,000,000,000 (octal).

2.2 MEMORY ADDRESSING

The 2100A Computer can be equipped with any one of six memory configurations, from 4K to 32K (K = 1024 words). The available configurations, which determine the addressing range, are: 4K, 8K, 12K, 16K, 24K, and 32K.

2.2.1 PAGING

The computer memory is logically divided into pages of 1024 words each. A page is defined as the largest block of memory which can be directly addressed by the memory address bits of a memory reference instruction (single-length). These memory reference instructions have 10 bits to specify a memory address, and thus the page size is 1024 locations (2000 in octal notation). Octal addresses for each page, up to the maximum memory size, are given in table 2.1.

Provision is made to address directly one of two pages: page zero (the base page, consisting of locations 00000 through 01777), and the current page (the page in which the instruction itself is located.) Memory reference instructions include a bit (bit 10) reserved to specify one or the other of these two pages. To address locations

MEMORY SIZE	PAGE	OCTAL ADDRESSES
	0	00000 to 01777
	1	02000 to 03777
	2	04000 to 05777
4K ↓	3	06000 to 07777
	4	10000 to 11777
	5	12000 to 13777
	6	14000 to 15777
8K ↓	7	16000 to 17777
	8	20000 to 21777
	9	22000 to 23777
	10	24000 to 25777
12K ↓	11	26000 to 27777
	12	30000 to 31777
	13	32000 to 33777
	14	34000 to 35777
16K ↓	15	36000 to 37777
	16	40000 to 41777
	17	42000 to 43777
	18	44000 to 45777
	19	46000 to 47777
	20	50000 to 51777
	21	52000 to 53777
	22	54000 to 55777
24K ↓	23	56000 to 57777
	24	60000 to 61777
	25	62000 to 63777
	26	64000 to 65777
	27	66000 to 67777
	28	70000 to 71777
	29	72000 to 73777
	30	74000 to 75777
32K ↓	31	76000 to 77777

Table 2.1. Memory Pages

in any other page, indirect addressing is used. Page references are specified by bit 10 as follows:

Logic 0 = Page Zero (Z)
Logic 1 = Current Page (C)

2.2.2 INDIRECT ADDRESSING

All memory reference instructions reserve a bit to specify direct or indirect addressing. For single-length memory reference instructions, bit 15 of the instruction word is used; for extended arithmetic memory reference instructions, bit 15 of the address word is used. Indirect addressing uses the address part of the instruction to access another word in memory, which is taken as a new memory reference for the same instruction. This new address word is a full 16 bits long, 15 bits of address plus another direct-indirect bit. The 15-bit length of the address permits access to any location in memory. If bit 15 again specifies indirect addressing, still another address is obtained; this multiple-step indirect addressing may be done to any number of levels. The first address obtained in the indirect phase which does not specify another indirect level becomes the effective address for the instruction. Direct or indirect addressing is specified by bit 15 as follows:

Logic 0 = Direct
Logic 1 = Indirect

2.2.3 RESERVED LOCATIONS

The first 64 memory locations of the base page (octal addresses 00000 through 00077) are reserved as listed below. The first two addresses are the A and B flip-flop register addresses and are not considered as core storage locations. (The actual corresponding core locations can, however, be loaded and read via the operator panel.) Locations 4 through 77 are reserved in the sense that interrupt wiring is present for the priority order given. As long as the locations do not have actual interrupt assignments (as determined by the input/output devices included in the user's system), these locations may be used for program purposes.

00000	Address of A-register
00001	Address of B-register
00002 00003	For exit sequence if A and B contents are used as executable words
00004	Interrupt location, highest priority (reserved for power fail interrupts)
00005	Reserved for memory parity and memory protect interrupts
00006	Reserved for direct memory access
00007	Reserved for direct memory access
00010 thru 00077	Interrupt locations in decreasing order of priority

The last 64 locations of memory (any size) are reserved for the basic binary loader. The basic binary loader is a permanently resident program to permit loading of binary information from punched paper tape (or disc, etc.) into memory. Unless specifically enabled by a panel switch, the loader locations are protected so they may not be altered or used in any way.

2.2.4 NONEXISTENT MEMORY

Nonexistent memory is defined as those memory locations not physically implemented in the machine (up to the maximum of 32K) and the last 64 locations of implemented memory when not enabled from the front panel. Any attempt to write into nonexistent memory will be ignored (no operation). Any attempt to read from a non-existent memory location will return an all-zero word; no parity error occurs.

2.3 HARDWARE REGISTERS

The 2100A Computer has six 16-bit working registers, two one-bit registers, and (on the operator panel) one 16-bit display register. The functions of these registers are described as follows:

M-REGISTER. The M-register holds the address of the memory cell currently being read from or written into.

T-REGISTER (MEMORY DATA). All data transferred into or out of memory is routed through the memory data register. When displayed, the display indicates the contents of the memory location currently pointed to by the M-register. The displayed data will go back into that location when any other action is taken (such as displaying some other register or beginning a run operation).

P-REGISTER. The P-register holds the address of the next instruction to be fetched out of memory. Since this is a "look-ahead" register, the P-register value will frequently differ from the M-register value. Table 2.2 lists P- and M-register contents for each of five different computer states, assuming the computer is halted.

A-REGISTER. The A-register is an accumulator, holding the results of arithmetic and logical operations performed by programmed instructions. This register may be addressed by any memory reference instruction as location 00000, thus permitting inter-register operation such as "add B to A," "compare B with A," etc., using a single-word instruction.

COMPUTER STATUS	P-REGISTER contains address of	M-REGISTER contains address of
FETCH	Current instruction	Last memory access
INDIRECT (after FETCH)	Current instruction	Current instruction
INDIRECT (after INDIRECT)	Current instruction	Last memory access
EXECUTE (after FETCH)	Next instruction	Current instruction
EXECUTE (after INDIRECT)	Next instruction	Last memory access

Table 2.2. P- and M-Register Indications

B-REGISTER. The B-register is a second accumulator, which can hold the results of arithmetic operations completely independent of the A-register. The B-register may be addressed by any memory reference instruction as location 00001 for inter-register operation with A.

S-REGISTER. The switch (S) register is a 16-bit utility register. In the halt mode, it may be manually loaded via the display register. In the run mode it may be addressed as in I/O device (select code 01) and receive and read back data to and from the accumulators.

EXTEND. The extend bit (E) is a one-bit register, and is used to link the A- and B-registers by rotate instructions or to indicate a carry from bit 15 of the A- or B-registers by an add instruction (ADA, ADB) or increment instruction (INA or INB, but not ISZ) which references these registers. This is of significance primarily for multiple-precision arithmetic. If already set, the extend bit is not complemented by a carry. It may be set, cleared, complemented, or tested by program instruction. The extend bit is set when the EXTEND light is on ("1") and clear when off ("0").

OVERFLOW. The overflow bit is a one-bit register which indicates that an add instruction (ADA, ADB), divide instruction (DIV), or an increment instruction (INA or INB, but not ISZ) referencing the A- and B-registers has caused (or will cause) the accumulators to exceed the maximum positive or negative number which they can contain. By program instructions, the overflow bit may be cleared, set, or tested. The OVF light remains on until the bit is cleared by an instruction and is not complemented if a second overflow occurs before being cleared. It will not be set by any shift or rotate instructions, except ASL (refer to definition in Section III).

DISPLAY REGISTER. The display register is included on the standard operator panel. It provides a means of displaying and modifying the contents of any of the six 16-bit working registers when the computer is in the halt mode. Each pushbutton is illuminated to indicate a content of "1," and is non-illuminated to indicate a content of "0." Each time a pushbutton is pressed, the content changes state. When the computer is in the run mode, the display register permanently displays the S-register contents.

2.4 INSTRUCTION FORMATS

Instructions for the 2100A Computer are classified according to format. The five formats used are illustrated in figure 2.2 and are described as follows. In all cases where a single bit is used to select one of two cases (e.g., D/I), the choice is made by coding a logic 0 or 1 respectively (i.e., 0/1).

MEMORY REFERENCE. This class of instructions combines an instruction code and a memory address into one word. This type of instruction is therefore used to execute some function involving data in a specific memory location. Examples are storing, retrieving, and combining memory data to or from the accumulators, or causing the program to jump to the specified location.

The cell referenced (i.e., the absolute address) is determined by a combination of the ten memory address bits in the instruction word (0 through 9) and five bits (10 through 14) assumed from the current condition of the P-register. This means that memory reference instructions can directly address any word in the current page; additionally, if the instruction is given in some location other than the base page (page zero), bit 10 of the instruction word doubles the addressing range to 2048 words by allowing selection of either page zero or current page. (This causes bits 10 through 14 of the address in the M-register to be reset to zero, instead of assuming the current indication of the P-register.) This feature provides a convenient linkage between all pages of memory, since page zero can be reached directly from any other page.

As discussed earlier, bit 15 is used to specify direct or indirect addressing. Also note that since the A- and B-registers can be addressed, any single-word memory reference instruction can apply to either of these registers as well as to memory cells. For example, ADA 0001 means add the contents of the B-register (its address being 0001) to the A-register; specify page zero for these operations, since the A- and B-register addresses are on page zero.

REGISTER REFERENCE. These instructions, in general, manipulate bits in the A-, B-, and E-registers. There is no reference to memory. This type includes 39 basic instructions, which are combinable to form a one-word multiple instruction that can operate in various ways on the contents of the A-, B- or E-registers. The

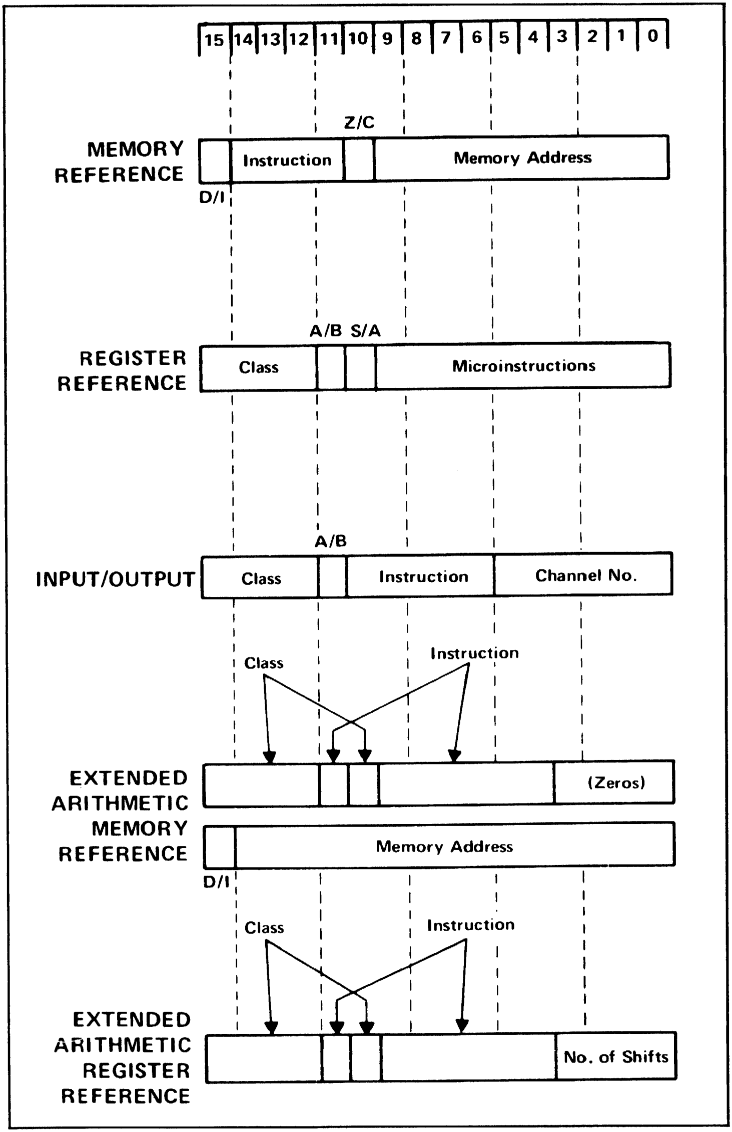


Figure 2.2 Instruction Formats

39 instructions are divided into two subgroups, the shift-rotate group (SRG) and the alter-skip group (ASG). These subgroups are specified by bit 10. Typical operations are clear and/or complement a register, conditional skips, and register increment.

INPUT/OUTPUT. The input/output class of instructions uses bits 6 through 11 for a variety of I/O instructions, and bits 0 through 5 to apply the instruction to a specific I/O channel. This provides a means of controlling all devices connected to the I/O channels, and for transferring data in or out. Also included in this group are instructions to control the interrupt system, overflow bit, and computer halt.

EXTENDED ARITHMETIC MEMORY REFERENCE. Like the single-word memory reference instruction above, the complete instruction includes an instruction code and a memory address. In this case, however, two words are required. The first word specifies the extended arithmetic class (bits 12 through 15 and 10) and the instruction code bits 4 through 9 and 11). Bits 0 through 3 are not needed and are coded with zeros. The second word specifies the memory address of the operand. Since a full 15 bits are used for the address, this type of instruction may directly address any location in memory. As with all memory reference instructions, bit 15 may be used to specify indirect addressing. Operations provided by this class of instructions are integer multiply and divide (using double-length product and dividend), and double load and double store.

EXTENDED ARITHMETIC REGISTER REFERENCE. This class of instructions provides long shifts and rotates on the combined A- and B-registers. Bits 12 through 15 and 10 identify the extended arithmetic class, and bits 4 through 9 and 11 specify the direction and type of shift. Bits 0 through 3 are used to specify the number of shifts, which can range from 1 to 16 places.

2.5 INTERRUPT SYSTEM

The computer interrupt system has 60 distinct interrupt levels. Each level has a unique priority assigned to it, and is associated with a numerically corresponding interrupt location in core memory.

As an example of the simplicity of this system: a service request from I/O channel 13 will cause an interrupt to core location 00013. The request for service will be granted on a priority basis higher than channel 14 but lower than channel 12. Thus a transfer in progress via channel 14 would be suspended to let channel 13 proceed, but a transfer via channel 12 could not be interrupted by channel 13.

Under program control, any device may be selectively enabled or disabled, thus switching the device in or out of the interrupt structure. In addition the entire interrupt system may be enabled or disabled under program control using a single instruction (excepting power fail and parity error interrupts).

Of the 60 interrupt levels, the two highest priority levels are reserved for hardware faults (power fail and parity error), the next two are reserved for DMA completion interrupts, and the remaining 56 are available for the I/O device channels. Table 2.3 lists interrupt levels in order of priority. Note that interrupt facilities for I/O channels above 25 (octal) are available through use of an I/O extender or multiplexer.

CHANNEL (Octal)	INTERRUPT LOCATION	ASSIGNMENT
04	00004	Power Fail Interrupt
05	00005	Memory Parity/Protect Interrupt
06	00006	DMA Channel 1 Completion Interrupt
07	00007	DMA Channel 2 Completion Interrupt
10	00010	I/O Device, highest priority
thru 25	00025	I/O Device (Mainframe)
thru 65	00065	I/O Device (Extender)
thru 77	00077	I/O Device (Multiplexer)

Table 2.3. Interrupt Assignments

Interrupt requests received while the computer is in halt mode will be processed, in order of priority, when the computer is put into run mode or is stepped single cycle.

2.5.1 POWER FAIL INTERRUPT

The computer is equipped with power sensing circuits. When primary power to the computer fails or drops below a safe operating level while the computer is running, an interrupt to memory location 00004 is automatically generated. This interrupt is given the highest priority in the system, and cannot be turned off or disabled. Location 00004 is intended to contain a jump-to-subroutine instruction referencing the entry point of a shut-down program, but it may alternatively contain a HLT instruction. Interrupt capability for lower-priority functions is automatically inhibited while a power fail routine is in progress. Sufficient time is available between the detection of power failure and the loss of usable internal power to execute about 100 instructions. The shut-down program should be written to save the current state of the computer system in memory, and then must halt the computer. A sample program is given in table 2.4.

Since the restoration of power might be unattended by an operator, the user is given a switch-selectable option of what action the computer should take. With the switch set to the halt position, the computer will halt when power is restored, whether the computer was running or halted when the failure occurred. (No panel indication is given.) With the switch in the restart position, the automatic restart feature is enabled. After a built-in delay of about a second following return to normal power levels, another interrupt is generated, again to location 00004. This time the shut-down portion of the subroutine is skipped (see sample subroutine) and the power-up portion begins. If the computer was not running when the power failure occurred, the computer is halted. If the computer was running, the system conditions are restored and the computer continues operation from the point of interruption. Alternatively, if location 00004 contains a HLT instead of a jump to a subroutine, the computer will halt at this time and EXTERNAL PRESET (or PRESET on the controller panel) will light.

To allow for the possibility of a second power failure occurring while the power-up routine is in progress, the user should limit the combined total of instructions (for both shut-down and power-up)

LABEL	OPCODE	OPERAND	COMMENTS
PFAR	NOP		Power fail/Auto Restart Subroutine
	SFC	4B	Skip if interrupt was caused by a power failure
	JMP	UP	Power is being restored, reset state of computer system
DOWN	STA	SAVA	Save A-register contents
	CCA		Set switch indicating that the computer was running when power failed
	STA	SAVR	
	STB	SAVB	Save B-register contents
	ERA,ALS		Transfer E-register content to A-register bit 15
	SOC		Increment A-register if Overflow is set
	INA		
	STA	SAVEO	Save E- and O-register contents
	LDA	PFAR	Save contents of P-register at time of power failure
	STA	SAVP	
	LIA	1B	Save contents of S-register
	STA	SAVS	
	⋮		Insert user-written routine to save I/O device states
	CLC	4B	Turn on restart logic so computer will restart when power is restored after momentary power failure
	HLT		Shutdown
UP	LDA	SAVR	Was computer running when power failed?
	SZA,RSS		
	JMP	HALT	No
	CLA		Yes, reset computer Run switch to initial state
	STA	SAVR	
	LDA	FENCE	Restore the memory protect fence register contents
	OTA	5B	
	⋮		Insert user-written routine to restore I/O device states
	LDA	SAVEO	Restore the contents of the E-register and O-register
	CLO		
	SLA,ELA		
	STF	1B	
	LDA	SAVS	Restore the contents of the S-register
	OTA	1B	
	LDA	SAVA	Restore A-register contents
	LDB	SAVB	Restore B-register contents
	STC	4B	Reset power fail logic for next power failure
	STC	5B	Turn on memory protect
	JMP	SAVP,I	Transfer control to program in execution at time of power failure
HALT	HLT		Return computer to halt mode
FENCE	OCT	2000B	Fence address storage (must be updated each time fence is changed)
SAVEO	OCT	0	Storage for E and O
SAVA	OCT	0	Storage for A
SAVB	OCT	0	Storage for B
SAVS	OCT	0	Storage for S
SAVP	OCT	0	Storage for P
SAVR	OCT	0	Storage for Run switch

Table 2.4. Sample Power Fail Subroutine

to less than 100. If the computer memory does not contain a subroutine to service the interrupt, location 00004 should contain a HLT 04 instruction (octal 102004).

A set control command (STC 04) must be given at the end of any restart routine. This command re-initializes the power fail logic and restores interrupt capability to lower priority functions. The EXTERNAL PRESET switch, when pressed, issues a similar command.

2.5.2 PARITY ERROR INTERRUPT

Parity checking of memory is a standard feature of the 2100A Computer. The parity logic continuously generates correct parity for all words written into memory and monitors the parity of all words read out of memory. Correct parity is defined as having the total number of "1" bits in a 17-bit memory word equal to odd value. If a "1" bit (or any odd number of "1" bits) is either dropped or added in the transfer process, a parity error signal is generated when the word is read out. Unless the error logic is specifically disabled by a CLF 05 instruction, the error signal causes an interrupt to location 00005.

Optionally (switch-selectable) the error signal may cause a halt, rather than an interrupt. The lighting of the HALT and PARITY indicators signals the fact that the halt was caused by a parity error. The PARITY light stays on until INTERNAL PRESET is pressed.

Assuming that the interrupt option is selected, the interrupt to location 00005 directs the computer to the entry point of a parity error subroutine. It is the user's decision as to what to do about a parity error; for example, he may want to record the address of the error location, or abort a critical operation. In any case, the PARITY light is turned off as soon as the interrupt is acknowledged and normal operation may be resumed on exit from the subroutine. An STF 05 instruction should be given at the end of the subroutine to re-initialize the logic.

In conjunction with the memory protect feature, it is possible to determine the address of the error location. The error address will automatically be loaded into the violation register of the memory protect logic, and from there it is accessible to the programmer. (See following discussion of memory protect interrupt.)

It is recommended on discovery of a parity error, that the entire program or set of data containing the error location be reloaded. However, knowing the address and contents of the error location, the user may be able to determine what operations have taken place as a result of reading the erroneous word. For example, if the word was an instruction, several other locations may be affected. By individually checking and correcting the contents of all affected locations, the user may resume running his program without a complete reload. If software is being generated, this may also need to be corrected.

2.5.3 MEMORY PROTECT INTERRUPT

Memory protect for the 2100A Computer is a standard feature. With this capability a selected block of memory of any size, from a settable fence address downward, is protected against alteration by memory reference instructions (excluding A- and B-register addresses, which may be freely addressed by any memory reference instruction except JMP). Also, when enabled, it prohibits the execution of all I/O instructions except those referencing I/O address 01 switch and overflow registers. This second feature limits the control of input/output operations to interrupt control only. Then, by programming the system to direct all I/O interrupts to an executive program in protected memory, the executive program can have exclusive control of the I/O system.

The memory protect logic is disabled by any interrupt (except if the interrupt location contains an input/output group instruction) and is re-enabled by an STC 05 instruction at the end of each interrupt subroutine. In the halt mode, memory protect is also disabled by the INTERNAL PRESET switch.

Programming rules pertaining to the use of memory protect, assuming the logic is enabled, are as follows:

a. Location 00002 is the lower boundary of protected memory. (Locations 00000 and 00001 are the A- and B-register addresses.)

b. JMP instructions may not reference the A- or B-registers. JSB, however, may do so.

c. The upper boundary is loaded into the fence register from the A- or B-registers by an OTA or OTB instruction with select code 05. Memory locations below (but not including) this address are protected.

d. Execution will be inhibited and an interrupt to location 00005 will occur if a JMP, JSB, ISZ, STA, STB, or DST instruction directly or indirectly addresses a location in protected memory, or if any I/O instruction is attempted (including halt, but excluding those addressing select code 01, the S- and overflow registers).

e. Any instruction not mentioned in "d" is legal, even if it does reference protected memory. In addition, indirect addressing through protected memory by those memory reference instructions listed in "d" is legal, provided the final effective address is outside protected memory.

Following a memory protect interrupt, the address of the illegal instruction will be present in the violation register. This address is made accessible to the programmer by an LIA 05 or LIB 05 instruction, which loads the address into the A- or B-register.

Since parity error and memory protect share the same interrupt locations, it is necessary to distinguish which type of error is responsible for the interrupt. If, after the LIA/B 05 instruction (preceding paragraph), bit 15 of the A-/B-register is a "1," parity error is indicated; if bit 15 is a "0," memory protect violation is indicated. In either case, the remaining bits of the register give the address of the error location.

Table 2.5 illustrates a sample memory protect and parity error subroutine. An assumption made for this example is that the location following the error location is an appropriate return point. This may not always be the case; for example, it may be advisable to abort the program in progress and return to a supervisory program.

LABEL	OPCODE	OPERAND	COMMENTS
MPPE	NOP		Memory Protect/Parity Error Subroutine
	CLF	0	Turn off interrupt system to inhibit I/O devices
	CLF	5	Turn off P.E. interrupt during subroutine
	STA	SVA	Save A-register contents
	STB	SVB	Save B-register contents
	LIA	5	Get contents of violation register in MP logic
	SSA		Check bit 15 to determine kind of error
	JMP	PERR	If a 1, go to parity error routine
	JMP	MPTR	If a 0, go to memory protect routine
MPTR	—		User's routine in case of memory protect violation
	—		
	—		
	etc.		
	—		
	—		
	—		
	LDA	SVA	Restore A-register
	LDB	SVB	Restore B-register
	STF	0	Enable interrupt system
	STF	5	Turn on parity error interrupt
	STC	5	Turn on memory protect interrupt
	JMP	MPPE,I	Exit the subroutine
PERR	—		User's routine in case of parity error
	—		
	—		
	etc.		
	—		
	—		
	—		
	JMP	PERR-6	Restore accumulators, turn on interrupts, exit

Table 2.5. Sample Memory Protect/Parity Error Subroutine

2.5.4 INTERRUPTS

The direct memory access (DMA) option provides high speed block transfers of data between I/O devices and memory. For the most part, DMA operates independently of the interrupt system. (Refer to the description of DMA operation in the Input/Output Section of this manual.)

The only time that DMA generates an interrupt is when it has completed transferring a specified block of data. Since there are two DMA channels, two interrupt locations are reserved for this option: location 00006 (interrupt from DMA channel 1) and location 00007 (interrupt from DMA channel 2). The channel 1 interrupt has priority over the channel 2 interrupt. Since these interrupts are primarily completion signals to the programmer and are therefore application dependent, no subroutine example is given.

2.5.5 I/O INTERRUPTS

The remaining interrupt locations (octal 00010 through 00077) are available to I/O devices. This represents a total of 56 (decimal) locations, one for each of 56 I/O channels.

In typical input/output operation, the computer issues a programmed command (e.g., set control/clear flag instruction STC,C) to one or more external devices, causing these devices to begin their read or write operation. Each device will put data into (input) or take data from (output) the input/output buffer on each individual interface card. During this time, the computer may continue running a program or may be programmed into a waiting loop to wait for a specific device. On completion of the read or write operation, each device returns an operation completed signal (flag) to the computer. The flags are passed through a priority network which allows only one device to be serviced regardless of the number of flags simultaneously present. The flag with the highest priority generates an interrupt signal at the end of the current machine cycle, except under any of the following circumstances.

a. Interrupt system disabled or device interrupt disabled.

b. JMP indirect or JSB indirect not sufficiently executed. These instructions inhibit all interrupts, except memory protect, until the instruction (plus one phase of the succeeding instruction is completed, or until at least three indirect references have occurred. The memory protect interrupt for a jump violation will occur on completion of the execute phase, but the jump itself will be inhibited.

c. Instruction in an interrupt location not sufficiently executed, even if of lower priority. Any interrupt inhibits the entire interrupt system until at least two phases have been completed. (JMP indirect and JSB indirect will be fully executed.)

d. Direct memory access option in process of transferring data.

e. The current instruction is one which may affect the priorities of input/output devices (STC, CLC, STF, CLF). The interrupt in this case must wait until the end of the succeeding machine cycle.

A set flag flip-flop inhibits all interrupt requests below it on the priority string (provided that the control flip-flop is also set). Once the flag flip-flop is cleared the next lower device can then interrupt. A service subroutine for any device can be interrupted only by a higher priority device; then, after the higher device is serviced, the interrupted subroutine may continue. In this way, it is possible for several service subroutines to be in a state of interruption at one time; each will be permitted to continue when the higher priority device is serviced. All service subroutines normally end with a JMP indirect instruction to return the computer to the point of interrupt.

For the programmer, communication with I/O devices is simplified by the availability of standard driver routines. Hewlett-Packard furnishes an I/O driver as an accessory to each standard peripheral device supplied by HP. The drivers supplied by HP conform to the design specifications of the HP Basic Control System and are subsequently referred to as BCS drivers. BCS drivers can be integrated into an existing basic control system simply by adding the additional driver to the system in a simple configuration process. BCS drivers generally have the following characteristics:

a. I/O is overlapped with processing using the computer priority interrupt system.

b. Each driver may operate identical devices occupying different I/O locations.

c. Provide status and error information to user and system I/O requests.

d. Compatible with other modules of HP software such as the Input/Output Control (IOC) program and the FORTRAN I/O program called the Formatter.

e. The object code for a BCS driver is relocatable binary.

The modularity of the basic control system provides the user with a very flexible operating system. The functions of the modules can be illustrated by following the sequence of events through a series of I/O transfers. An input transfer is used as an example. See Figure 2.3.

The user or system I/O request is made to a unique entry point in the IOC program. After checking the request for validity, IOC obtains the memory address of the BCS driver for the requested device. Control is transferred to the BCS driver and the input operation is initiated. After initiation the BCS driver transfers control back to the user or system program. The program continues processing until the I/O device completes a single operation. At that time an interrupt request is generated, which forces transfer of control to the BCS driver once again. The data is transferred between the device and a specified memory buffer and the I/O device is commanded to do another operation. This process continues until all data has been transferred and the user or system input request is satisfied.

The equipment table (EQT) is a memory table created at configuration time to describe the hardware I/O channel of the device, the name and address of the I/O driver to be used, a status word, and a transmission log to be used by the I/O driver. Each physical I/O device (or, sometimes, I/O subsystem consisting of two or more devices) in the system is defined by an entry in the EQT. The EQT provides the interface between IOC and the BCS driver and in addition provides for device independent programming.

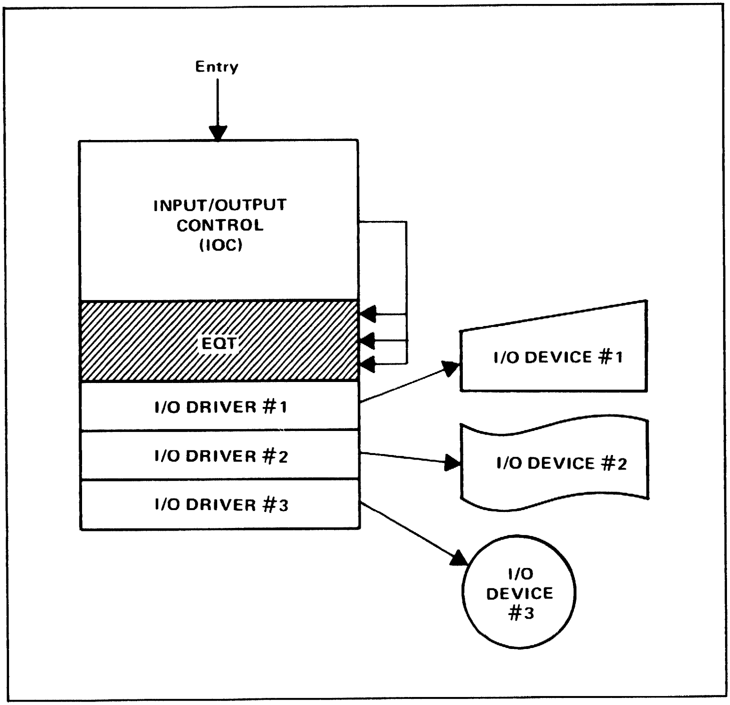


Figure 2.3. Modules of BCS

2.5.6 INTERRUPT REGISTER

Each time an interrupt occurs, the address of the interrupt location is stored in the central interrupt register. The contents of this register is accessible at any time with an LIA 04 or LIB 04 instruction. This puts the address of the most recent interrupt into the a- or B-register.

2.5.7 INTERRUPT SYSTEM CONTROL

I/O address 00 is a master control address for the interrupt system. An STF 00 instruction enables the entire interrupt system, and a CLF 00 instruction disables the interrupt system. The two exceptions are the power fail interrupt, which cannot be disabled, and parity error interrupt, which can only be selectively enabled or disabled by STF 05 or CLF 05, respectively.

Whenever power is turned on, a clear signal to I/O address 00 automatically disables the interrupt system. The INTERRUPT SYSTEM pushbutton on the operator panel may be used to switch the interrupt system on or off manually. However, programs dependent on interrupt operation should include an STF 00 instruction to ensure that the interrupt system is enabled in the run mode.

INSTRUCTIONS 3

This section defines each of the 80 machine instructions of the 2100A Computer. Definitions are grouped according to instruction type: memory reference, register reference, input/output, extended arithmetic memory reference, and extended arithmetic register reference.

With each definition is a diagram showing the machine coding of the instruction. The light shaded bits code the instruction type and the dark shaded bits code the specific instruction. Unshaded bits are further described under the introduction to each instruction type. The mnemonic code and instruction name are given above each diagram.

In all cases where an additional bit is used to specify a secondary function (D/I, Z/C, or H/C), the choice is made by coding a logic 0 or 1 respectively. That is, a logic 0 codes D, Z, and H, and a logic 1 codes I, C, and C. These abbreviations are defined as follows:

D	=	Direct addressing
I	=	Indirect addressing
Z	=	Zero page
C	=	Current page
H	=	Hold flag
C	=	Clear flag

3.1 INSTRUCTION TIMING

All instructions except ISZ and the extended arithmetic instructions are fully executed in 1.96 microseconds. ISZ is executed in 2.94 microseconds, and the extended arithmetic instructions are executed in the times shown in table 3.1. The Divide instruction executes faster than shown if the divisor is positive (15.68 microseconds) or if overflow occurs (11.76 microseconds). If indirect addressing is used with any of the single-word memory reference instructions, 0.98 microsecond is added for each level of indirect addressing used; 1.96 microseconds are added for each level of indirect addressing with extended arithmetic memory reference instructions.

Instructions are executed in two or more phases. The first phase is the fetch phase, which obtains an instruction from memory and transfers it into the central processor's instruction register. Next, there can be one or more indirect phases. The indirect phase, which applies only to single-length memory reference instructions, obtains a new operand address for the same (current) instruction.

INSTRUCTION		TIME (μsec)
MPY (Multiply)		10.78
DIV (Divide)	Max	16.66
DLD (Double Load)		5.88
DST (Double Store)		5.88
	Number of Shifts	
ASR	1, 2, 3	2.94
(Arithmetic	4, 5, 6, 7, 8	3.92
Shift	9, 10, 11, 12, 13	4.90
Right)	14, 15, 16	5.88
ASL	1, 2, 3, 4, 5	4.90
(Arithmetic	6, 7, 8, 9, 10	5.88
Shift	11, 12, 13, 14, 15	6.86
Left)	16	7.84
LSR, RRR	1, 2	2.94
(Logical	3, 4, 5, 6, 7	3.92
Shift Right,	8, 9, 10, 11, 12	4.90
Rotate Right)	13, 14, 15, 16	5.88
LSL, RRL	1, 2, 3, 4	4.90
(Logical	5, 6, 7, 8, 9	5.88
Shift Left,	10, 11, 12, 13, 14	6.86
Rotate Left)	15, 16	7.84

Table 3.1. Extended Arithmetic Execution Times

Lastly, there is an execute phase, which accomplishes actual execution of the instruction. For extended arithmetic memory reference instructions, indirect addressing is also accomplished in the execute phase. Although the duration of a phase varies considerably (from 588 nanoseconds to an indeterminate time in the case of extended arithmetic indirect addressing), synchronization with memory or input/output operations results in overall execution times as specified in the preceding paragraph.

3.2 MEMORY REFERENCE INSTRUCTIONS

The 14 memory reference instructions execute a function involving data in memory. Bits 0 through 9 specify the affected memory location on a given memory page or, if indirect addressing is used, the next address to be referenced. Indirect addressing may be continued to any number of levels; when the D/I bit is "0" (specifying direct addressing), that location will be taken as the effective address. The A- and B-registers may be addressed as locations 00000 and 00001 (octal) respectively.

In bit 10 (Z/C) is a "0," the memory address is on page zero; if bit 10 is a "1," the memory address is on the current page. If the A- or B-register is addressed, bit 10 must be a "0" to specify page zero, unless the current page is page zero.

AND "AND" TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	0	1	0	Z/C										

Memory Address

The contents of the addressed location is logically "anded" to the contents of the A-register. The contents of the memory is left unaltered.

JSB JUMP TO SUBROUTINE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	0	1	1	Z/C										

Memory Address

This instruction, executed in location P, causes computer control to jump unconditionally to the memory location (m) specified in the address portion of the JSB instruction word. The contents of the P-register plus one (return address) is stored in location m, and the next instruction to be executed will be that contained in the next location (m + 1). A return to the main program sequence at P + 1 may be effected by a jump indirect through location m.

XOR "EXCLUSIVE OR" TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	1	0	0	Z/C										

Memory Address (bits 9–0)

The contents of the addressed location is combined with the contents of the A-register as an "exclusive or" logic operation. The contents of the memory is left unaltered.

JMP JUMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	1	0	1	Z/C										

Memory Address (bits 9–0)

The instruction transfers control to the addressed location. That is, JMP causes the P-register to be set according to the memory address portion of the instruction word, so that the next instruction will be read from that location.

IOR "INCLUSIVE OR" TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	1	1	0	Z/C										

Memory Address (bits 9–0)

The contents of the addressed location is combined with the contents of the A-register as an "inclusive or" logic operation. The contents of the memory cell is left unaltered.

ISZ INCREMENT AND SKIP IF ZERO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	1	1	1	Z/C										

Memory Address

An ISZ instruction adds one to the contents of the addressed memory location. If the result of this operation is zero, the next instruction is skipped; i.e., the P-register is advanced by two instead of one. Otherwise, the program proceeds normally to the next instruction in sequence. The incremented value is written back into the memory cell in either case. An ISZ instruction referencing locations zero or one (A- or B-register) cannot cause setting of the extend or overflow bits (unlike INA and INB).

ADA ADD TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	0	0	0	Z/C										

Memory Address

The contents of the addressed memory location is added to the contents of the A-register, and the sum remains in the A-register. The result of the addition may set the extend or overflow bits. The contents of the memory cell is unaltered.

ADB ADD TO B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	0	0	1	Z/C										

Memory Address

The contents of the addressed memory location is added to the contents of the B-register, and the sum remains in the B-register. Extend or overflow bits may be set, as for ADA. The contents of the memory cell is unaltered.

CPA COMPARE TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	0	1	0	Z/C										

Memory Address (bits 9–0)

The contents of the addressed location is compared with the contents of the A-register. If the two 16-bit words are unequal, the next instruction is skipped; i.e., the P-register is advanced by two instead of one. If the words are identical, the program proceeds normally to the next instruction in sequence. The contents of neither the A-register nor the memory cell is altered.

CPB COMPARE TO B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	0	1	1	Z/C										

Memory Address (bits 9–0)

Same as CPA, except comparison is made with the B-register.

LDA LOAD A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	1	0	0	Z/C										

Memory Address (bits 9–0)

The A-register is loaded with the contents of the addressed location. The contents of the memory cell is unaltered.

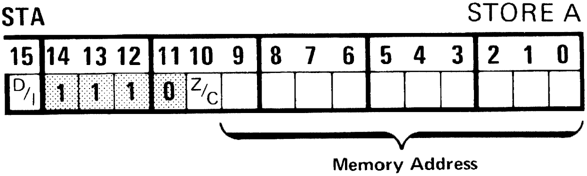
LDB LOAD B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	1	0	1	Z/C										

Memory Address (bits 9–0)

The B-register is loaded with the contents of the addressed location. The contents of the memory cell is unaltered.

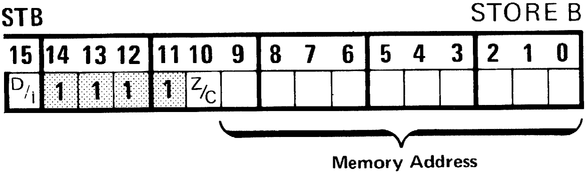
STA STORE A



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	1	1	0	Z/C										

The contents of the A-register is stored in the addressed location. The previous contents of the memory cell is lost; the A-register is unaltered.

STB STORE B



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	1	1	1	1	Z/C										

The contents of the B-register is stored in the addressed location. The previous contents of the memory cell is lost; the B-register is unaltered.

3.3 REGISTER REFERENCE INSTRUCTIONS

The 39 register reference instructions execute various functions on data contained in the A-, B-, and E-registers. The instructions are divided into two groups: the shift-rotate group and the alter-skip group. In each group, several instructions may be combined into one word and are thus individually termed microinstructions. Since the two groups are separate and distinct, microinstructions from the two groups cannot be mixed. Unshaded bits in the coding diagrams are available for combining other microinstructions.

SHIFT-ROTATE GROUP. The 20 instructions of the shift-rotate group are defined first. A comparison of shift and rotate functions is given in figure 3.1. Rules for combining microinstructions are as follows. (Refer to table 3.2.)

a. Only one microinstruction can be chosen from the multiple-choice columns.

b. References to A- and B-registers cannot be mixed.

c. The sequence of execution is left to right.

d. In machine code, use zeros to exclude unwanted micro-instruction bits.

e. Use a “1” bit in bit 9 to enable shifts or rotates in the first position, and a “1” bit in bit 4 to enable shifts or rotates in the second position.

f. The extend bit is not affected unless specifically stated. However, if a rotate-with-E instruction (ERA/B, ELA/B) is coded but disabled by a “0” in bit 9 or 4, the E-register will be updated even though the A- or B-register is not affected; code a NOP (three zeros) to avoid this situation.

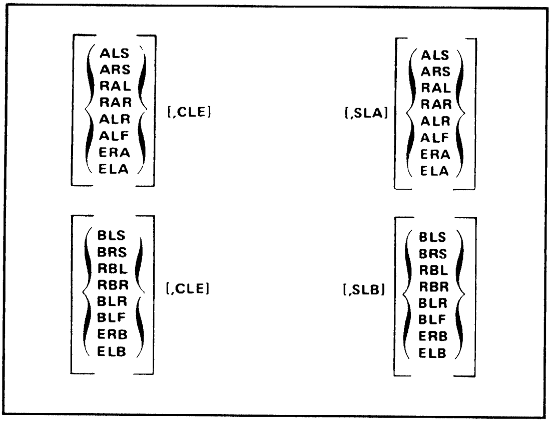


Table 3.2. Shift-Rotate Combining Guide

NOP NO OPERATION

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

An all-zero instruction word causes a no-operation cycle.

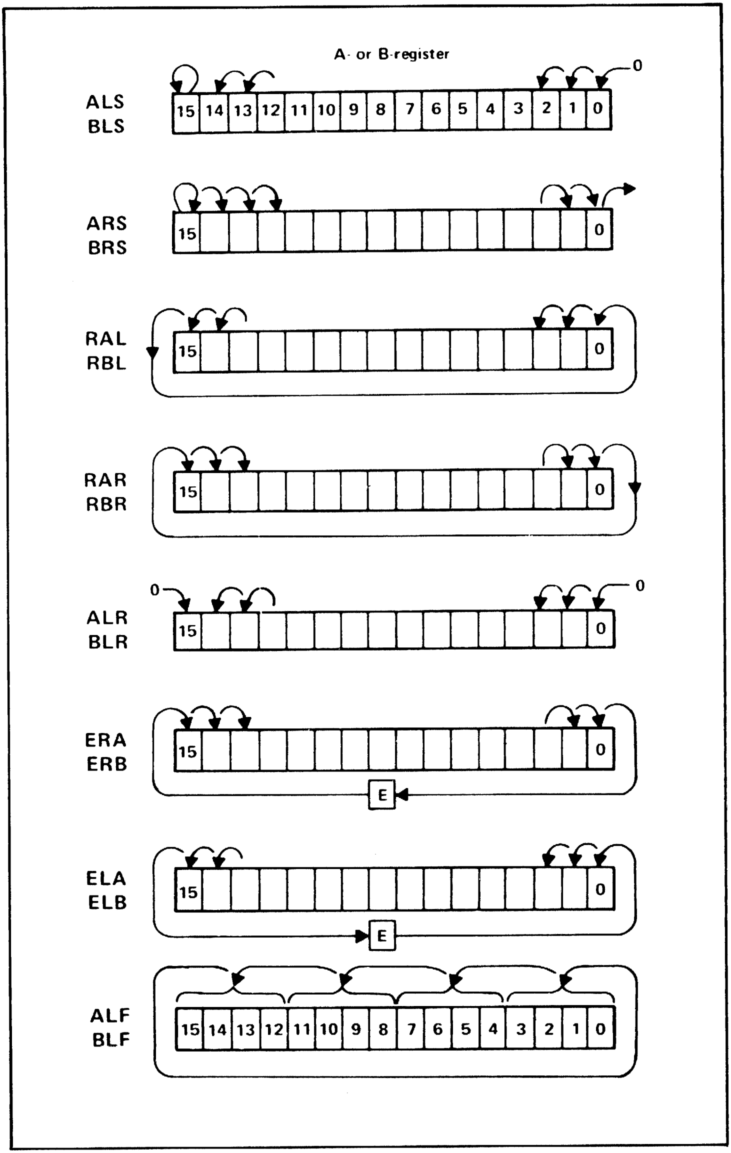


Figure 3.1. Shift and Rotate Functions

CLE CLEAR E

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

Clear E-register (extend bit).

SLA SKIP IF LSB OF A IS ZERO

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

The next instruction is skipped if the least significant bit of the A-register is "0."

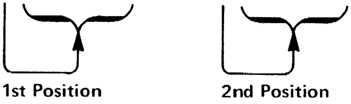
SLB SKIP IF LSB OF B IS ZERO

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

The next instruction is skipped if the least significant bit of the B-register is "0."

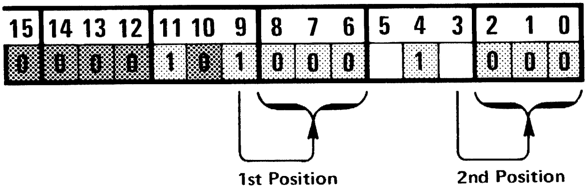
ALS A LEFT SHIFT

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



The A-register is arithmetically shifted left one place, 15 magnitude bits only. Bit 15 (sign bit) is not affected; bit shifted out of bit 14 is lost. A "0" replaces vacated bit 0.

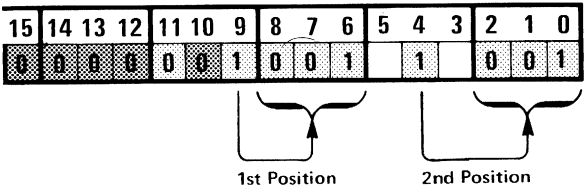
BLS B LEFT SHIFT



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

The B-register is arithmetically shifted left one place, 15 magnitude bits only. Bit 15 (sign bit) is not affected; bit shifted out of bit 14 is lost. A "0" replaces vacated bit 0.

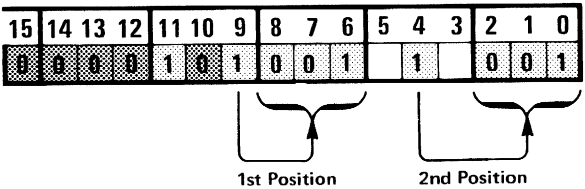
ARS A RIGHT SHIFT



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

The A-register is arithmetically shifted right one place, 15 magnitude bits only. Bit 15 (sign bit) is not affected; copy of sign bit is shifted into bit 14. Bit shifted out of bit 0 is lost.

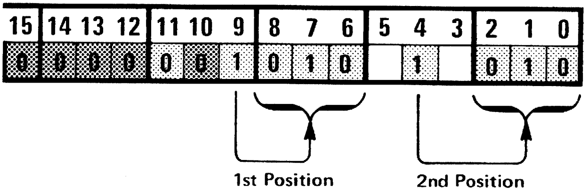
BRS B RIGHT SHIFT



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

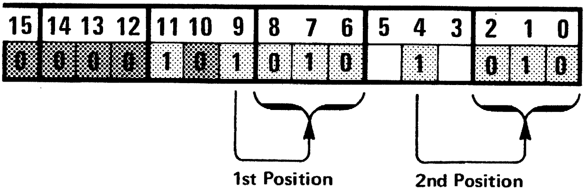
The B-register is arithmetically shifted right one place, 15 magnitude bits only. Bit 15 (sign bit) is not affected; copy of sign bit is shifted into bit 14. Bit shifted out of bit 0 is lost.

RAL ROTATE A LEFT



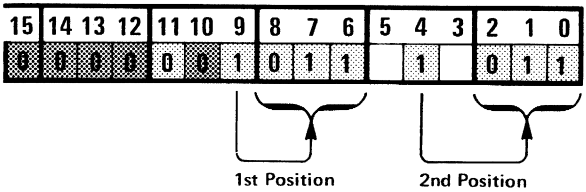
Rotate A-register left one place, all 16 bits. Bit 15 is rotated around to bit 0.

RBL ROTATE B LEFT



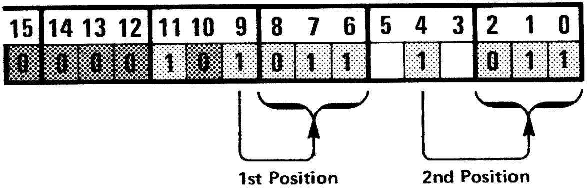
Rotate B-register left one place, all 16 bits. Bit 15 is rotated around to bit 0.

RAR ROTATE A RIGHT



Rotate A-register right one place, all 16 bits. Bit 0 is rotated around to bit 15.

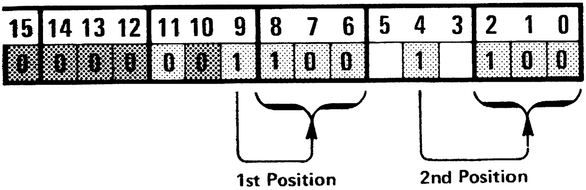
RBR ROTATE B RIGHT



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

Rotate B-register right one place, all 16 bits. Bit 0 is rotated around to bit 15.

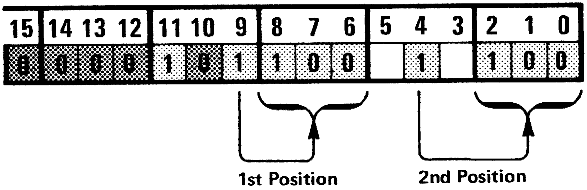
ALR A LEFT SHIFT, CLEAR SIGN



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

Shift A-register left one place, same as ALS, but clear sign bit after shift.

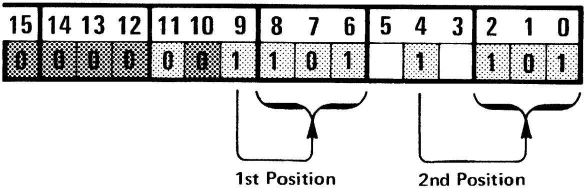
BLR B LEFT SHIFT, CLEAR SIGN



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

Shift B-register left one place, same as BLS, but clear sign bit after shift.

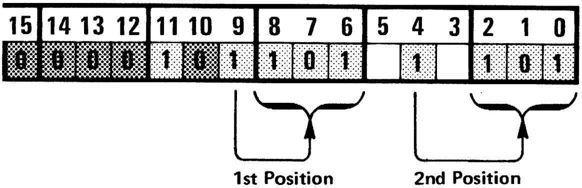
ERA ROTATE E RIGHT WITH A



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

Rotate E-register right with A-register, one place (17 bits). Bit 0 is rotated into extend register; extend contents is rotated into bit 15.

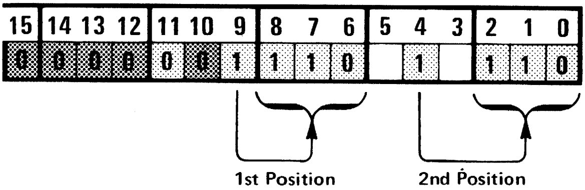
ERB ROTATE E RIGHT WITH B



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

Rotate E-register right with B-register, one place (17 bits). Bit 0 is rotated into extend register; extend contents is rotated into bit 15.

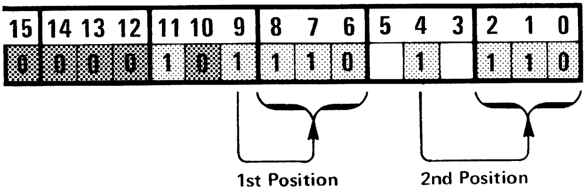
ELA ROTATE E LEFT WITH A



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

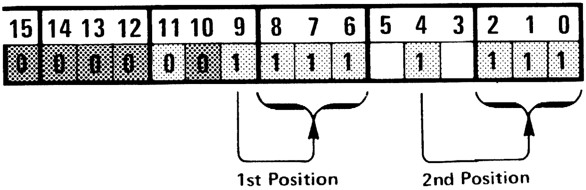
Rotate E-register left with A-register, one place (17 bits). Bit 15 is rotated into extend register; extend contents is rotated into bit 0.

ELB ROTATE E LEFT WITH B



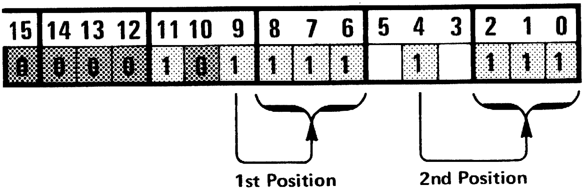
Rotate E-register left with B-register, one place (17 bits). Bit 15 is rotated into extend register; extend contents is rotated into bit 0.

ALF ROTATE A LEFT FOUR



Rotate A-register left four places, all 16 bits. Bits 15, 14, 13, 12 are rotated around to bits 3, 2, 1, 0 respectively. Equivalent to four successive RAL instructions.

BLF ROTATE B LEFT FOUR



Rotate B-register left four places, all 16 bits. Bits 15, 14, 13, 12 are rotated around to bits 3, 2, 1, 0 respectively. Equivalent to four successive RBL instructions.

ALTER-SKIP GROUP. The 19 instructions of the alter-skip group are defined next. This group is specified by a "1" bit in bit 10.

Rules for combining microinstructions are as follows. (Refer to table 3.3).

a. Only one microinstruction can be chosen from the multi-choice columns.

b. References to A- and B-registers cannot be mixed.

c. The sequence of execution is left to right.

d. If two or more skip functions are combined, the skip will occur if either or both conditions are met. One exception exists: refer to RSS instruction.

e. In machine code, use zeros to exclude unwanted micro instruction bits.

$$\left[\begin{Bmatrix}\text{CLA}\\ \text{CMA}\\ \text{CCA}\end{Bmatrix}\right]\ \text{[,SEZ]}\ \left[\begin{Bmatrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{Bmatrix}\right]\ \text{[,SSA] [,SLA] [,INA] [,SZA] [,RSS]}$$

$$\left[\begin{Bmatrix}\text{CLB}\\ \text{CMB}\\ \text{CCB}\end{Bmatrix}\right]\ \text{[,SEZ]}\ \left[\begin{Bmatrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{Bmatrix}\right]\ \text{[,SSB] [,SLB] [,INB] [,SZB] [,RSS]}$$

Table 3.3. Alter-Skip Combining Guide

CLA CLEAR A

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

Clear the A-register

CLB CLEAR B

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

Clear the B-register

CMA COMPLEMENT A

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

Complement the A-register (One's complement.)

CMB COMPLEMENT B

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

Complement the B-register (One's complement.)

CCA CLEAR AND COMPLEMENT A

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

Clear, then complement the A-register. Puts 16 one's in the A-register; this is the two's complement form of -1.

CCB CLEAR AND COMPLEMENT B

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

Clear, then complement the B-register. Puts 16 one's in the B-register; this is the two's complement form of -1.

CLE CLEAR E

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

Clear the E-register (extend bit).

CME COMPLEMENT E

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

Complement the E-register (extend bit).

CCE CLEAR AND COMPLEMENT E

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

Clear, then complement the E-register (extend bit). Sets the extend bit to "1."

SEZ SKIP IF E IS ZERO

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

Skip the next instruction if the E-register (extend bit) is zero.

SSA SKIP IF SIGN OF A IS ZERO

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

Skip next instruction if the sign bit (bit 15) of the A-register is zero; i.e., skip if the contents of A is positive.

SSB SKIP IF SIGN OF B IS ZERO

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

Skip next instruction if the sign bit (bit 15) of the B-register is zero; i.e., skip if the contents of B is positive.

SLA SKIP IF LSB OF A IS ZERO

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

Skip next instruction if the least significant bit of the A-register is zero; i.e., skip if an even number is in A.

SLB SKIP IF LSB OF B IS ZERO

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

Skip next instruction if the least significant bit of the B-register is zero; i.e, skip if an even number is in B.

INA INCREMENT A

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

Increment the A-register by one. Can cause setting of extend or overflow bits.

INB INCREMENT B

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

Increment the B-register by one. Can cause setting of extend or overflow bits.

SZA SKIP IF A IS ZERO

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

Skip next instruction if the A-register is zero (16 zeros).

SZB SKIP IF B IS ZERO

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

Skip next instruction if B-register is zero (16 zeros).

RSS REVERSE SKIP SENSE

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

Skip occurs for any of the preceding skip instructions, if present, when the non-zero condition is met. RSS without a skip instruction in the word causes an unconditional skip. If a word with RSS also includes both SSA/B and SLA/B bits 15 and 0 must both be one for skip to occur. In all other cases the skip occurs if one or more skip condition is met.

3.4 INPUT/OUTPUT INSTRUCTIONS

The 17 input/output instructions provide the capability to set or clear the I/O flag and control bits and the overflow bit, to test the state of the overflow and I/O flag bits, and to transfer data between an I/O channel and the A- and B-registers. In addition, specific instructions in this group control the interrupt system and can cause a programmed halt.

Bit 11, where relevant, specifies the A- or B-register or distinguishes between set control and clear control; otherwise it may be “1” or “0” without affecting the instruction (although the assembler will assign zeros, as shown). Bit 9, where not specified, offers the choice of holding (0) or clearing (1) the device flag after execution of the instruction. (Exception: the H/C bit associated with the last two instructions in this list holds or clears the overflow bit instead of a flag bit.) Bits 8, 7, and 6 identify the instruction. Bits 5 through 0 (unshaded) form select codes to make the instruction apply to one of up to 64 input/output devices or functions.

HLT

HALT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0		1	H/C	0	0	0						

Select Code

Halts the computer and holds or clears the flag (according to bit 9) of any desired input/output device (bits 5 through 0). The HLT instruction has the same effect as the HALT pushbutton: the HALT switch lights, and the front-panel control switches are enabled. the HLT instruction will be displayed (MEMORY DATA is automatically selected when computer halts), and the P-register will normally indicate the halt location plus one.

STF

SET FLAG

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

Select Code

Sets the flag of the selected I/O channel or function. An STF 00 instruction enables the interrupt system for all select codes (except power fail and parity error, which are always enabled).

CLF

CLEAR FLAG

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

Select Code

Clears the flag of the selected I/O channel or function. A CLF 00 instruction disables the interrupt system for all select codes (except power fail and parity error, which are always enabled); this does not affect the status of the individual channel flags.

SFC SKIP IF FLAG CLEAR

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

Select Code

Skip next instruction if the flag of the selected channel is clear (device busy).

SFS SKIP IF FLAG SET

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

Select Code

Skip next instruction if the flag of the selected channel is set (device ready).

MIA MERGE INTO A

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

Select Code

The contents of the input/output buffer associated with the selected device is merged ("inclusive or") into the A-register.

MIB MERGE INTO B

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

Select Code

The contents of the input/output buffer associated with the selected device is merged ("inclusive or") into the B-register.

LIA LOAD INTO A

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

Select Code

The contents of the input/output buffer associated with the selected device is loaded into the A-register.

LIB LOAD INTO B

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

Select Code

The contents of the input/output buffer associated with the selected device is loaded into the B-register.

OTA OUTPUT A

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

Select Code

The contents of the A-register is loaded into the input/output buffer associated with the selected device. If the buffer is less than 16 bits in length, the least significant bits of the A-register normally are loaded. (Some exceptions exist, depending on the type of output device.) A-register contents is not altered.

OTB OUTPUT B

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

Select Code

The contents of the B-register is loaded into the input/output buffer associated with the selected device.

STC SET CONTROL

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

Select Code

Sets the control bit of the selected I/O channel or function.

CLC CLEAR CONTROL

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

Select Code

Clears the control bit of the selected I/O channel or function. This turns off a device channel and prevents it from interrupting. A CLC 00 instruction clears all control bits from select code 06 and up, effectively turning off all I/O devices.

STO SET OVERFLOW

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

Sets the overflow bit.

CLO — CLEAR OVERFLOW

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

Clears the overflow bit.

SOS — SKIP IF OVERFLOW SET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	H/C	0	1	1	0	0	0	0	0	1

If the overflow register is set, the next instruction of the program is skipped. Use of the H/C bit will hold or clear the overflow bit following execution of this instruction (whether the skip is taken or not).

SOC — SKIP IF OVERFLOW CLEAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	H/C	0	1	0	0	0	0	0	0	1

If the overflow register is clear, the next instruction of the program is skipped. Use of the H/C bit will hold or clear the overflow bit following execution of this instruction (whether the skip is taken or not).

3.5 EXTENDED ARITHMETIC MEMORY REFERENCE INSTRUCTIONS

The four extended arithmetic memory reference instructions provide for integer multiply and divide, and for loading and storing double-length words to and from the accumulators. The complete instruction requires two words: one for the instruction code, and one for the address. When stored in memory the instruction word is the first to be fetched; the address word is in the next higher location.

Since 15 bits are available for the address, these instructions may directly address any location in memory. As for all memory reference instructions, indirect addressing to any number of levels may also be used. A "0" in the D/I bit specifies direct addressing; a "1" specifies indirect addressing.

MPY MULTIPLY

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
D/I															

Memory Address

Multiplies a 16-bit integer in the A-register by a 16-bit integer in the addressed memory location. The resulting double-length integer product resides in the B- and A-registers, with the B-register containing the sign bit and most significant 15 bits of the quantity. The A-register may be used as an operand (i.e., memory address 0), resulting in an arithmetic square. Overflow cannot occur; the instruction clears the overflow bit.

DIV DIVIDE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
D/I															

Memory Address

Divides a doubleword integer in the combined B- and A-registers by a 16-bit integer in the addressed memory location. The result is a 16-bit integer quotient in the A-register and a 16-bit integer remainder in the B-register. Overflow can result from an attempt to divide by zero, or from an attempt to divide by a number too small for the dividend. In the former case (divide by zero) execution will be attempted with unpredictable results left in the B- and A-registers. In the latter case (divisor too small) the division will not be attempted and the B- and A-register contents will be unchanged, except that a negative quantity will be made positive. If there is no divide error, the overflow bit is cleared.

DLD DOUBLE LOAD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
D/I															

Memory Address

Loads the contents of addressed memory location m (and m+1) into the A- and B-registers, respectively.

DST DOUBLE STORE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
D/I															

Memory Address

Stores the doubleword quantity in the A- and B-registers into addressed memory locations m (and m+1), respectively.

3.6 EXTENDED ARITHMETIC REGISTER REFERENCE INSTRUCTIONS

The six extended arithmetic register reference instructions provide various types of shifting operations on the combined contents of the B- and A-registers. The B-register is considered to be on the left (most significant word) and the A-register is considered to be on the right (least significant word). An example of each type of shift operation is illustrated in figure 3.2.

The complete instruction is given in one word and includes four bits (unshaded) to specify the number of shifts, from 1 to 16. By viewing the four bits as a binary-coded number, the number of shifts is easily expressed; e.g., binary-coded 1 for one shift, binary-coded 2 for two shifts, etc. The maximum of 16 shifts is coded with four zeros; this essentially exchanges the B- and A-register contents.

The extend bit is not affected by any of the following instructions. Except for the arithmetic shifts, overflow also is not affected.

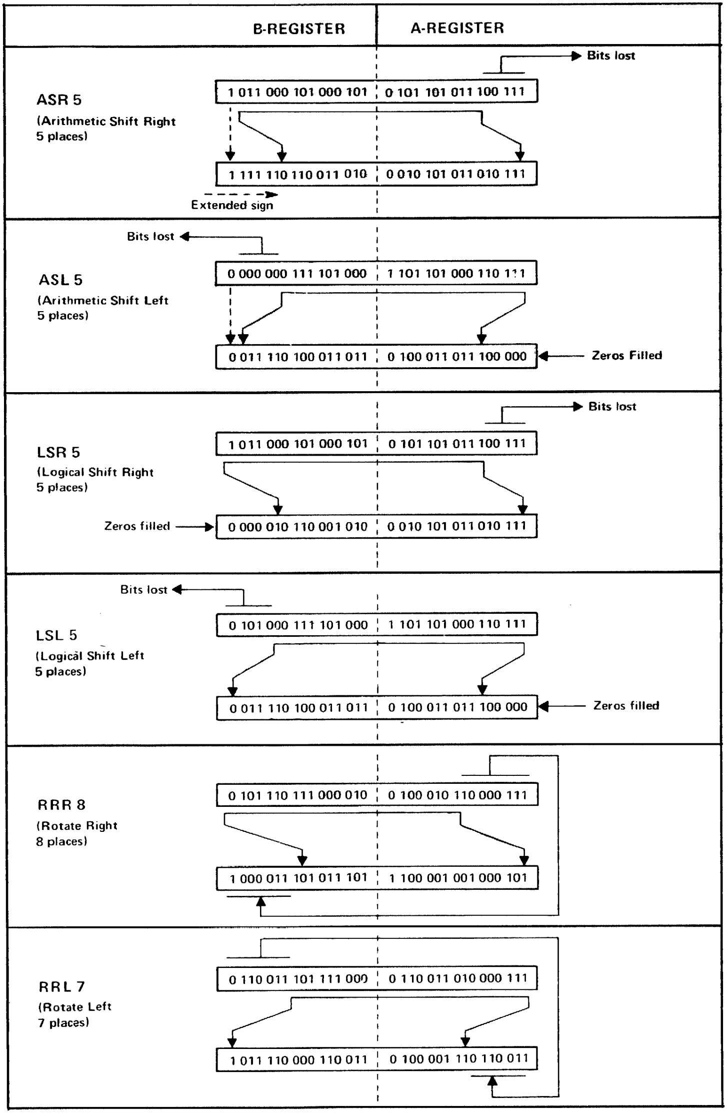


Figure 3.2. Examples of Doubleword Shifts and Rotates

ASR ARITHMETIC SHIFT RIGHT

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

Number of Shifts

Arithmetically shifts the combined contents of the B- and A-registers right, n places. The value of n may be any number from 1 through 16. The sign bit is unchanged and is extended into bit positions vacated by the right shift. Data bits shifted out of the least significant end of the A-register are lost. Overflow cannot occur; the instruction clears the overflow bit.

ASL ARITHMETIC SHIFT LEFT

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

Number of Shifts

Arithmetically shifts the combined contents of the B- and A-registers left, n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated low order positions of the A-register. The sign bit is unchanged, and data bits are lost out of bit 14 of the B-register. If one of the bits lost is a significant data bit ("1" for positive numbers, "0" for negative numbers), overflow will be set; otherwise, overflow will be cleared during execution. See ASL example in figure 3.2. (Note that two additional shifts in this example would cause an error by losing a significant "1.")

LSR LOGICAL SHIFT RIGHT

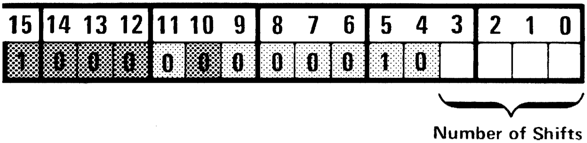
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	0	0	0	1	0				

Number of Shifts

Logically shifts the combined contents of the B- and A-registers right, n places. The value of n may be any number from 1 through

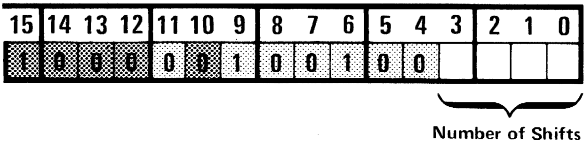
16. Zeros are filled into vacated high order bit positions of the B-register, and data bits are lost out of the low order bit positions of the B-register.

LSL LOGICAL SHIFT LEFT



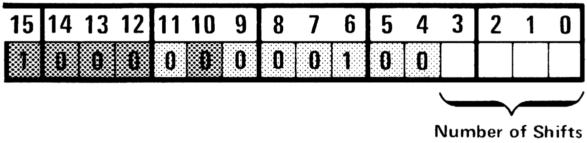
Logically shifts the combined contents of the B- and A-registers left, n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated low order bit positions of the A-register, and data bits are lost out of the high order bit positions of the B-register.

RRR ROTATE RIGHT



Rotates the combined contents of the B- and A-registers right, n places. The value of n may be any number from 1 through 16. No bits are lost or filled in. Data bits shifted out of the low order end of the A-register are rotated around to enter the high order end of the B-register.

RRL ROTATE LEFT

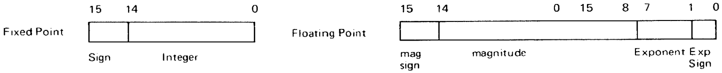


Rotates the combined contents of the B- and A-registers left, n places. The value of n may be any number from 1 through 16. No bits are lost or filled in. Data bits shifted out of the high order end of the B-register are rotated around to enter the low order end of the A-register.

3.7 FLOATING POINT INSTRUCTIONS (Optional)

Each of the six floating point instructions has a unique machine code associated with it. When a floating point instruction is assembled, the assembler places the appropriate machine code in the program. FORTRAN and ALGOL Compilers generate a subroutine call to the Program Library. The Library replaces the subroutine call with the appropriate machine code. Thus the Library is used only once. Execution of floating point machine code calls the appropriate firmware routine which allows the micro processor to execute the instruction. A complete summary of 2100 floating point instructions is given in table 3.4.

DATA FORMAT



INSTRUCTION:	ADD	SUBTRACT	MULTIPLY	DIVIDE	FIX	FLOAT
PURPOSE:	to add two floating point numbers, x and y	to subtract the floating point number y from the floating point number x	to multiply two floating point numbers, x and y	to divide the floating point number x by the floating point number y	to convert the floating point number x to integer format	to convert the integer i to floating point format
MACHINE CODE:	105000B	105020B	105040B	105060B	105100B	105120B
CALLING SEQUENCE:	FAD DEF Y[,I]	FSB DEF Y[,I]	FMP DEF Y[,I]	FDV DEF Y[,I]	FIX	FLT (i is assumed to be in the A register)
			(X is assumed to be in the A, B registers)			
ASSEMBLY LANGUAGE:	FAD Y	FSB Y	FMP Y	FDV Y	FIX	FLT (i is assumed to be in the A register)
			(X is assumed to be in the A, B registers)			
RETURN:		Floating point result is left in the A, B registers			Integer result is left in A register. Any fractional part is truncated. B register content is meaningless.	Floating point result is left in the A, B registers
MINIMUM EXECUTION TIME: (including Fetch)	23.52 μsec	24.50 μsec	33.32 μsec	51.94 μsec	5.88 μsec	9.80 μsec
MAXIMUM EXECUTION TIME: (including Fetch)	59.78 μsec	60.76 μsec	41.16 μsec	55.86 μsec	8.82 μsec	24.50 μsec
EXECUTION TIME FOR EACH LEVEL OF INDIRECT:	.98 μsec	.98 μsec	.98 μsec	.98 μsec	– – –	– – –
ERROR CONDITION:	If the result is outside the range of representable floating point numbers, $[-2^{127}, 2^{127}(1-2^{-23})]$, the overflow flag is set and the result $2^{128}(1-2^{-23})$ is returned.				If the magnitude of the floating point number is $\geq 2^{15}$, the integer 32767 (077777B) is returned and ovflo flag is set.	None
	If an underflow occurs, (result within the range $[-2^{-129}(1+2^{-22}), 2^{-129}]$), the overflow flag is set and the result 0 is returned.				If the magnitude of the floating point number is $\leqslant 1$, the integer 0 is returned.	

Table 3.4. Floating Point Instruction Specifications

INPUT/OUTPUT SYSTEM 4

The purpose of the input/output system is to transfer data between the computer and external devices.

Normally, data is transferred through the A- or B-register. Refer to figure 4.1. This type of transfer occurs in three distinct steps:

a. between external device and its interface card in the computer;

b. between the interface card and the A- or B-register; and

c. between the A- or B-register and memory.

This three-step process applies to both the "in" direction (as above) and the "out" direction (reverse order). This type of transfer, which is executed under program control, allows the computer logic to manipulate the data during the transfer process.

Data may also be transferred automatically under control of the direct memory access (DMA) option. Once the DMA option has been initialized, no programming is involved, and the transfer is reduced to a two-step process: the transfer between the device and its interface, and the transfer between the interface and memory. Two DMA channels are provided and are assignable to operate with any two device interfaces.

Since the DMA transfer eliminates programmed loading and storing via the accumulators, the time involved is very short. Thus DMA is used with high-speed devices capable of transferring data at rates up to 1,020,400 sixteen-bit words per second. Further information on the direct memory access option is given later in this section.

4.1 I/O ADDRESSING

As shown in figure 4.2, an external device is connected by a cable directly to an interface card located inside the computer. The

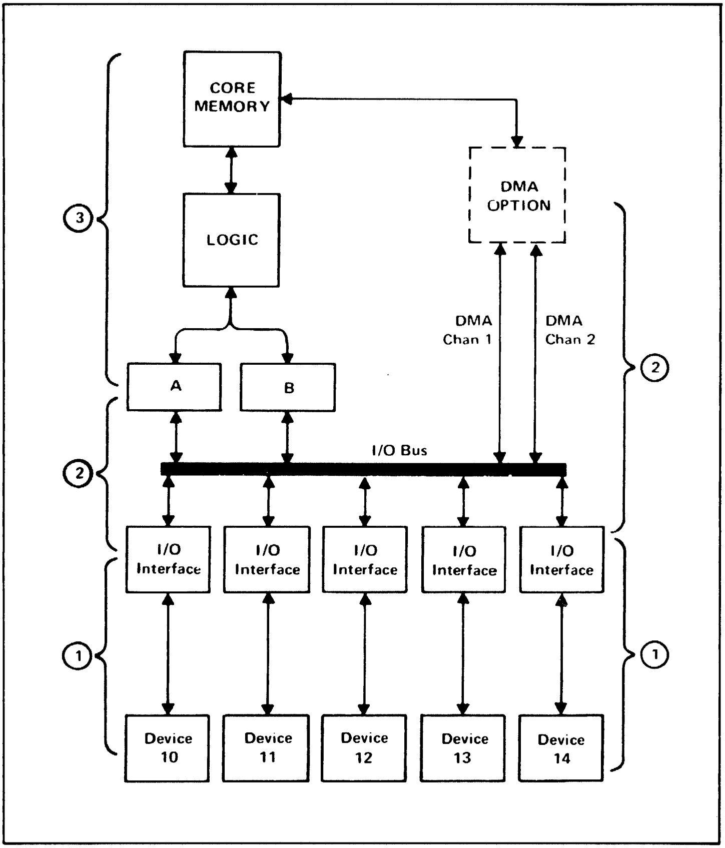


Figure 4.1. Input/Output System

interface card, in turn, plugs into one of the 14 input/output slots. Each slot is assigned a fixed address, called the select code. The computer can then communicate with a specific device on the basis of its select code.

Figure 4.2 shows an interface card being inserted into the I/O slot having the highest priority. This slot is assigned select code 10 (octal). If it is decided that the associated device should have lower priority, its interface and cable may be exchanged with those occupying some other I/O slot. This will change both the priority and the I/O address. However, due to priority chaining (explained later), there can be no vacant slots from select code 10 to the highest used select code (if the interrupt mode is to be used). Only select codes 10 through 77 (octal) are available for input/output devices. The lower select codes (00 through 07) are reserved for other features discussed elsewhere in this manual. As figure 4.2 shows, select codes 10 through 25 are available in the mainframe of the computer. If an I/O extender is used, slot 25 is used for interconnection of the extender, and select codes 25 through 65 will then be available in the extender. This is a total of 45 (decimal) select codes. The full range of 56 select codes may be plugged into any slot, but the rule that there can be no vacant slots (select codes) from 10 upward must be maintained.

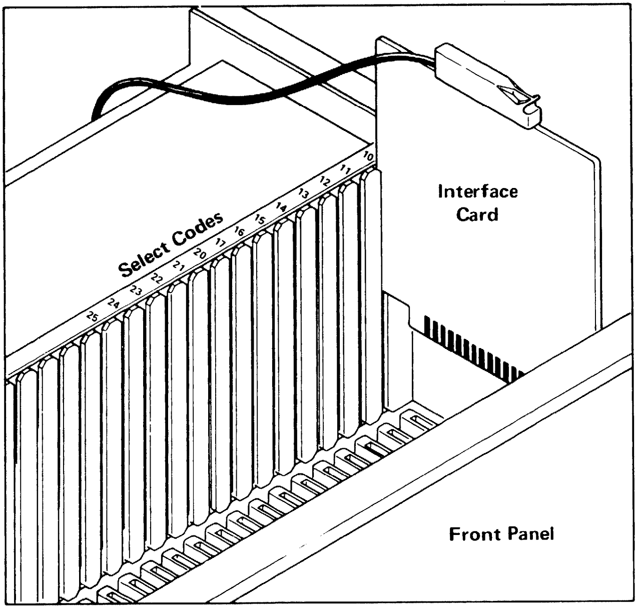


Figure 4.2. I/O Address Assignments

In some cases, certain devices may require two I/O slots and two select codes. This requirement is fully explained in documentation supplied with the applicable interface.

4.2 I/O PRIORITY

When a device is ready to be serviced (refer to "I/O Data Transfer"), it causes its interface to request an interrupt so that the computer will interrupt the current program and service the device. Since many device interfaces will be requesting service at random times, it is necessary to establish an orderly sequence for granting interrupts. Secondly, it is desirable that high-speed devices should not have to wait for low-speed device transfers.

Both of these requirements are met by a series-linked priority structure, illustrated in simplified form in figure 4.3. The bold line, representing a priority enabling signal, is routed in series through each card which is capable of causing an interrupt. The card may not interrupt unless this enabling signal is present at its input.

Each device (or other interrupt function) can break the enabling line when it requests an interrupt. If two devices simultaneously request an interrupt, obviously the device with the lowest select code number will be the first one which can interrupt, since it has broken the enable line for the higher select codes. The other device cannot begin its service routine until the first device is finished; however, a still higher priority device (lower select code) may interrupt the service routine of the first device.

Figure 4.4 illustrates a hypothetical case in which several devices require servicing by interrupting a CPU program. Both simultaneous and time-separate interrupt requests are considered.

Assume that the computer is running a CPU program when an interrupt from I/O channel 12 occurs (at reference time t1). A JSB instruction in the interrupt location for select code 12 causes a program jump to the service routine for the channel 12 device. The JSB instruction automatically saves the return address (in a location which the programmer must reserve in his routine) for a later return to the CPU program.

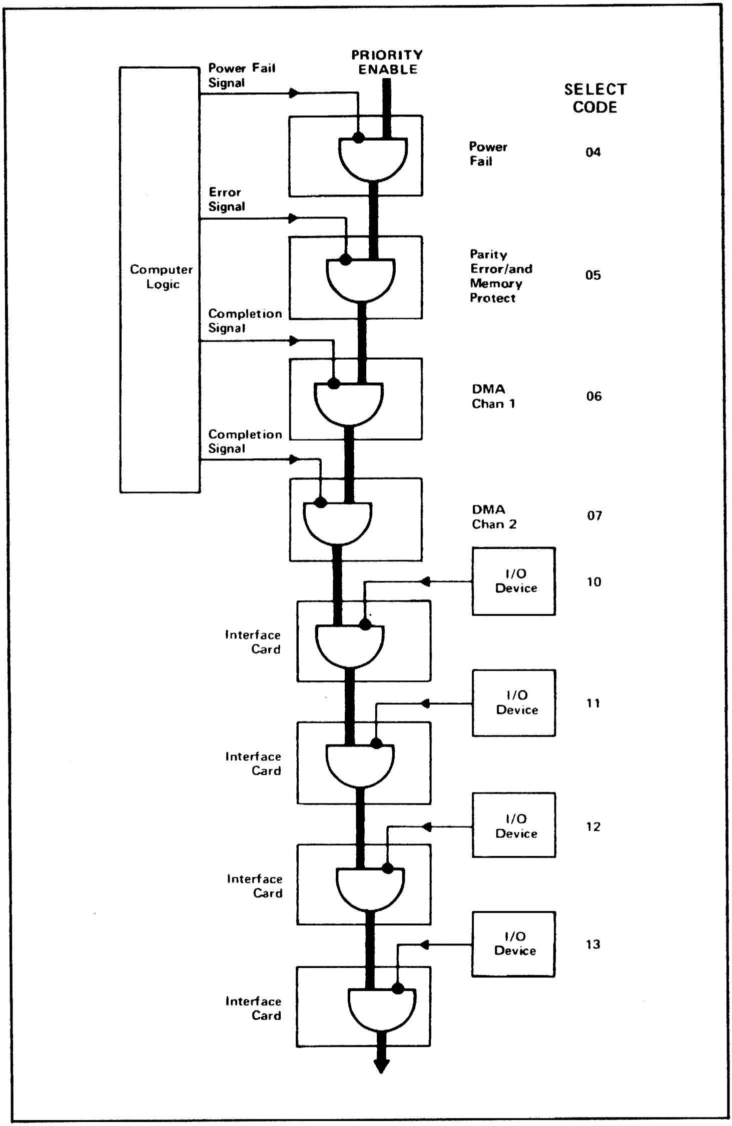


Figure 4.3. Priority Linkage

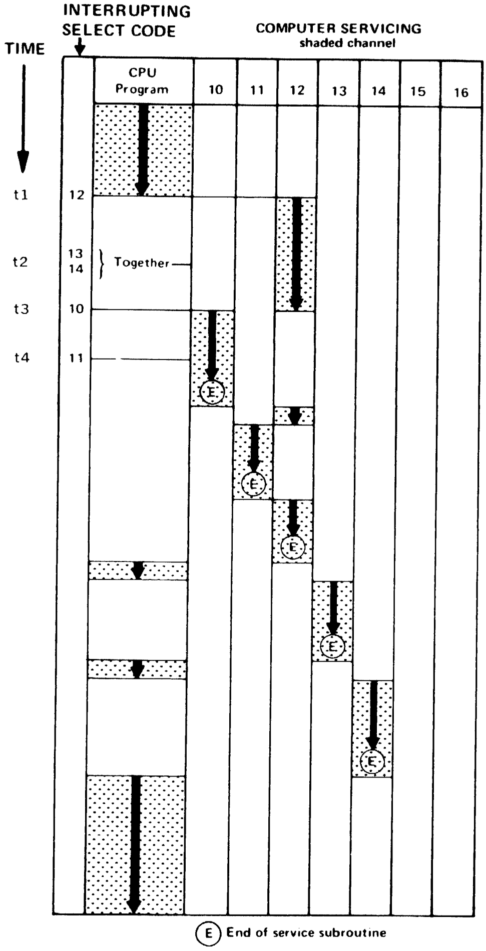


Figure 4.4. Interrupt Sequences

The routine for channel 12 is not completed when several other devices request service (set flag). First, channels 13 and 14 request simultaneously at t2; however, neither has priority over channel 12, so their flags are ignored and channel 12 continues its transfer. But at t3, a higher priority device on channel 10 requests service. This request interrupts the channel 12 transfer and causes the channel 10 transfer to begin. The JSB instruction saves the return address for return to the channel 12 routine.

During the channel 10 transfer, device 11 sets the channel 11 flag (t4). Since it has lower priority than channel 10, device 11 must wait until the end of the channel 10 routine. And since channel 10, when it ends, contains a return address to the channel 12 routine, program control temporarily returns to channel 12 (even though the waiting channel 11 has higher priority). The JMP,1 instruction used for the return inhibits all interrupts until fully executed (plus one phase of the next instruction). At the end of this short interval, the channel 11 interrupt request is granted.

When channel 11 has finished its routine, it returns control to channel 12, which at last has sufficient priority to complete its routine. Since channel 12 has been saving a return address in the main CPU program, it returns control to this point.

The two waiting interrupt requests from channels 13 and 14 are now enabled. Since channel 13 has higher priority, it goes first. At the end of its routine, it temporarily returns control to the CPU program. Then the lowest priority channel, 14, interrupts and completes its transfer. Finally, control is returned to the CPU program, which continues processing.

4.3 INTERFACE ELEMENTS

The interface card provides a communication link between the computer and an external device. There are three basic elements on the interface card which either the computer or device can control in order to effect the necessary communication. These elements are as follows:

4.3.1 CONTROL BIT

This is a one-bit flip-flop register used by the computer to turn on the device channel. When set, the control bit generates a start command to the device, telling it to begin one operation cycle (e.g., read or write one character or word). The interface cannot interrupt unless the control bit is set. The control bit is set by an STC (set control) instruction and cleared by a CLS (clear control) instruction, with a specific select code (e.g., STC 12 or CLC 12). The device cannot affect the control bit.

4.3.2 FLAG BIT

This is a one-bit flip-flop register primarily used by the device to indicate, when set, that transmission between the device and the interface buffer has been completed. Computer instructions can also set the flag (STF), clear the flag (CLF), test if it is set (SFS), and test if it is clear (SFC). The device cannot clear the flag bit. If the corresponding control bit is set, priority is high, and the interrupt system is enabled. Setting the flag bit will cause an interrupt to the location corresponding to the device's select code.

4.3.3 BUFFER

This is a flip-flop register for intermediate storage of data. Typically the data capacity is 8 or 16 bits, but this is entirely dependent on the type of device.

4.4 I/O DATA TRANSFER

The preceding paragraphs of this section have discussed the individual features of the I/O system. The following paragraphs show how data is actually transferred under interrupt control. The sequences are highly simplified in order to present an overall view, without the involvement of software operating systems and device drivers. For more detailed information refer to the documentation supplied with the appropriate software system or interface kit.

4.4.1 INPUT TRANSFER

The upper part of figure 4.5 illustrates the sequence of operations for an input transfer. Note that some of the operations are under control of the computer program (programmer's responsibility) and some of the operations are automatic. The sequence is as follows:

The operation begins with a programmed instruction to set control and clear flag on the addressed interface card (1). In this example it is assumed that the interface card is installed in the slot for select code 12; thus the instruction is STC 12,C. Since the next few operations are under automatic control of the hardware, the computer program may continue executing other instructions.

Setting the control bit causes the interface card to issue a start command (2) to the external device. The device then proceeds with its electromechanical process of reading a character. When it has done so, it sends a signal (done) back to the interface card, along with the data character (3).

At the interface card the "done" signals sets the flag bit. The flag, in turn, generates an interrupt (4)—provided the interrupt conditions previously mentioned are met. That is, the interrupt system must be on (STF 00 previously given), no higher priority interrupt may be requesting, and the control bit must be set (done in step 1).

The interrupt causes the current computer program to be suspended, and control is transferred to a service subroutine (5). It is the programmer's responsibility to provide the linkage between the interrupt location (00012 in this case) and the service subroutine. Also, it is the programmer's responsibility to include in his service subroutine the instructions for processing of the data (loading into an accumulator, manipulating if necessary, and storing into memory).

The subroutine may then issue further STC 12,C commands to transfer additional characters. One of the final instructions in the service subroutine must be a clear control (CLC 12 in this case). This step (6) allows lower priority devices to interrupt (equivalent to re-enabling a gate in figure 4.3) and restores the channel to its static "ready" condition—control cleared and flag set. This condition is initially established by the computer at turn-on, and it is the programmer's responsibility to return the channel to the same condition on the completion of each transfer.

At the end of the subroutine, control is returned to the interrupt program via previously established linkages.

4.4.2 OUTPUT TRANSFER

The lower part of figure 4.5 illustrates the sequence of operations for an output transfer. Again note the distinction between programmed and automatic operations.

It is assumed that the data to be transferred has been loaded into A-register and is in a form suitable for output. The interface card is assumed to be installed in the slot for select code 13.

The operation begins with a programmed instruction to transfer the data from the A-register to the interface buffer (1). The instruction in this example is OTA 13. This is followed (2) by an instruction to set control and clear flag; i.e., STC 13,C. Since the next few operations are under automatic control of the hardware, the computer program may continue executing other instructions.

Setting the control bit causes the interface card to read out the buffer data to the device and to issue a start command (3). The device proceeds to write the data, and when it has finished the device sends a signal (done) back to the interface card (4).

At the interface card the "done" signal sets the flag bit. The flag, in turn, generates an interrupt (5)—provided the interrupt system is on, priority is high, and the control bit is still set (from step 2).

The interrupt causes the current computer program to be suspended, and control is transferred to a service subroutine (6). It is the programmer's responsibility to provide the linkage between the interrupt location (00013 in this case) and the service subroutine. The detailed contents of the subroutine is also the programmer's responsibility, and will vary with the type of device.

The subroutine may then output further data to the interface card and re-issue the STC 13,C command for additional character transfers. One of the final instructions in the service subroutine must be a clear control (CLC 13). This step (7) allows lower priority

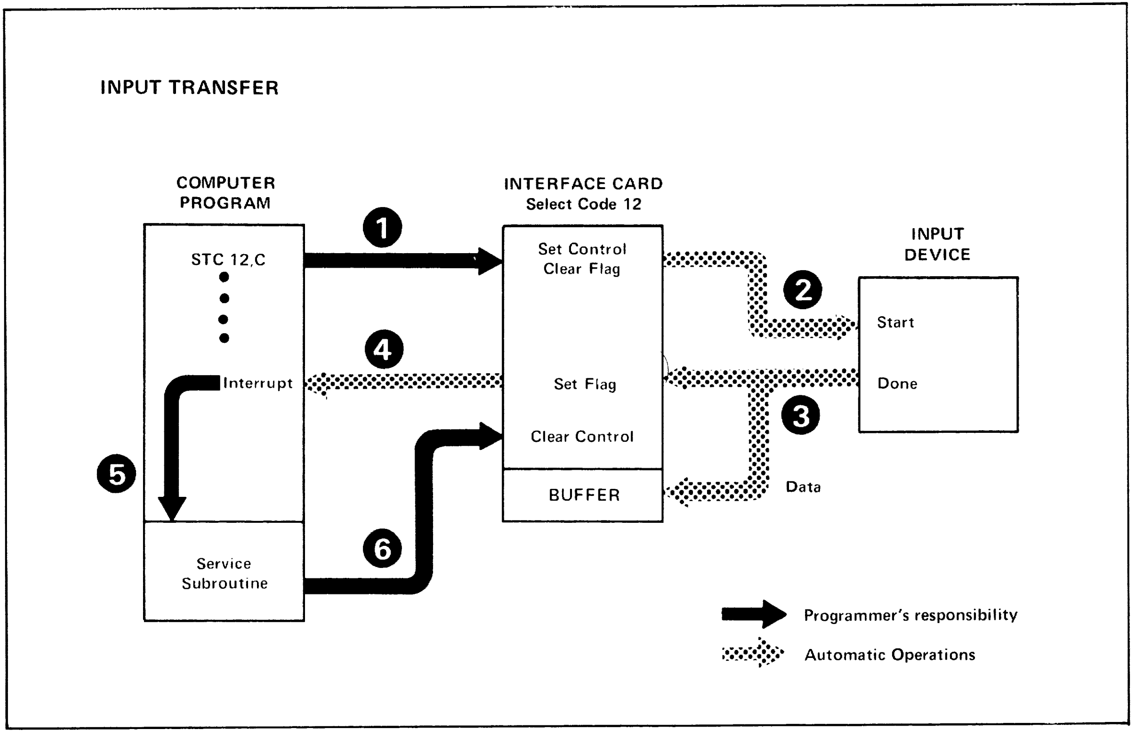


Figure 4.5. Input/Output Transfers (Part 1)

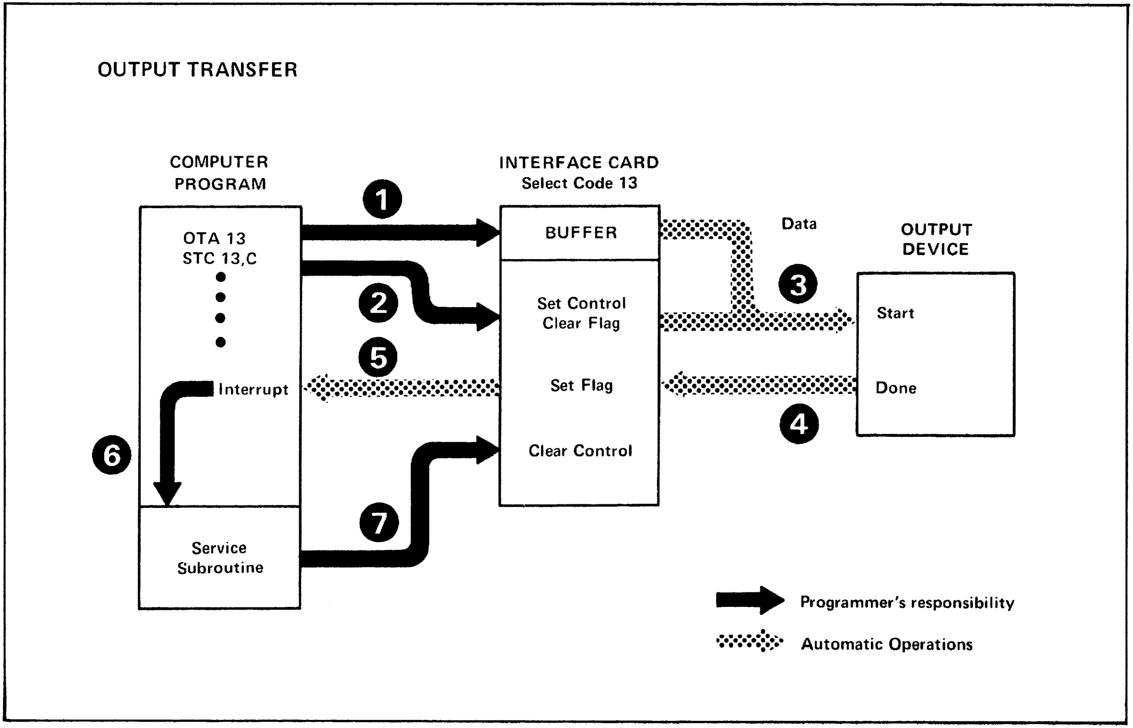


Figure 4.5. Input/Output Transfers (Part 2)

devices to interrupt, and restores the channel to its static "ready" condition—control cleared and flag set. At the end of the subroutine, control is returned to the interrupted program via previously established linkages.

4.4.3 NON-INTERRUPT TRANSFERS

It is also possible to transfer data without using the interrupt system. This involves a "wait-for-flag" method, in which the computer commands the device to operate and then waits for the completion response. It is therefore assumed that computer time is relatively unimportant. The programming is very simple, consisting of only four words of in-line coding, as shown in table 4.1. Each of these routines will transfer one word or character of data. It is assumed that the interrupt system is turned off (STF 00 not previously given).

INPUT

INSTRUCTIONS	COMMENTS
STC 12,C	Start device
SFS 12	Is input ready?
JMP *-1	No, repeat previous instruction
LIA 12	Yes, load input into A-register

OUTPUT

INSTRUCTIONS	COMMENTS
OTA 13	Output A-register to buffer
STC 13,C	Start device
SFS 13	Has device accepted the data?
JMP *-1	No, repeat previous instruction
NOP	Yes, proceed

Table 4.1. Non-Interrupt Transfer Routines

INPUT. As before, an STC 12,C instruction begins the operation by commanding the device to read one word or character. The computer then goes into a waiting loop, repeatedly checking the status of the flag bit. If the flag is not set, the JMP *-1 instruction causes a jump back to the SFS instruction. (The *-1 operand is assembler notation for "this location minus one.") When the flag is set, the skip condition for SFS is met and the JMP instruction is skipped. The computer thus exits from the waiting loop, and the LIA 12 instruction loads the device's input data into the A-register.

OUTPUT. The first step of output is to transfer the data to the interface buffer; the OTA 13 instruction does this. Then STC 13,C commands the device to operate and accept the data. The computer then goes into its waiting loop, the same as described in the preceding paragraph. When the flag is received, indicating that the device has accepted the output data, the computer exits from the loop. (The final NOP is for illustration purposes only.)

4.5 DIRECT MEMORY ACCESS

As indicated earlier in figure 4.1, the purpose of the direct memory access (DMA) option is to provide a direct data path, software assignable, between memory and a high-speed peripheral device.

DMA accomplishes this purpose by stealing a memory cycle instead of interrupting to a service subroutine. The DMA option for the 2100A Computer is capable of stealing every consecutive memory cycle, and thus can transfer data at rates up to 1,020,400 words per second.

There are two DMA channels, each of which may be separately assigned to operate with any I/O interface, including those in an HP 2155A I/O Extender. When both DMA channels are in simultaneous operation, channel 1 has priority over channel 2. The combined maximum transfer rate for both channels operating together is 1,020,400 words per second; the rate available to channel 2 is then the rate difference between 1,020,400 and channel 1's actual rate.

When DMA is accessing memory, it has priority over CPU access of memory. Thus the rate available to the CPU when DMA is operating is the difference between 1,020,400 words per second and the actual transfer rate of DMA channels 1 and 2 combined.

DMA transfers are on a full-word basis; hardware packing and unpacking of characters is not provided. The word count register is a full 16 bits in length.

DMA transfers are accomplished in blocks. The transfer is initiated by an initialization routine, and from then on operation is under automatic control of the hardware. The initialization routine tells DMA which direction to transfer the data (in or out), where in memory to put or take data, which I/O channel to use, and how much data to transfer. Completion of the block transfer is signalled by an interrupt to location 00006 (for channel 1) or location 00007 (for channel 2) if the interrupt system is enabled. It is also possible to check for completion by testing the status of the flag for select code (for channel 1) or select code 03 (for channel 2). A block transfer can be aborted with an STF 06 or 07 instruction.

4.5.1 DMA OPERATION

Figure 4.6 illustrates the sequence of operations for a DMA transfer. Comparison with conventional transfers (figure 4.5) shows that much more of the operation is automatic. Remember that the procedures in figure 4.5 must be repeated for each word or character. In figure 4.6 the automatic DMA operations will transfer a block of data of any size, limited only by the availability of memory space.

The sequence of events is as follows. (Input transfer is illustrated; the minor differences for output are explained in text.)

The initialization routine sets up the control registers on the DMA card (1) and issues the first start command (STC 12,C) directly to the interface card. (If the operation is output, the buffer is also loaded at this time.) The DMA option is then turned on and the computer program continues with other instructions.

Setting control and clearing flag on the interface card (2) causes a start command (3) to the external device (with data if output). The device goes through its read or write cycle and returns a "done" signal (4), with data if input. The set flag, regardless of

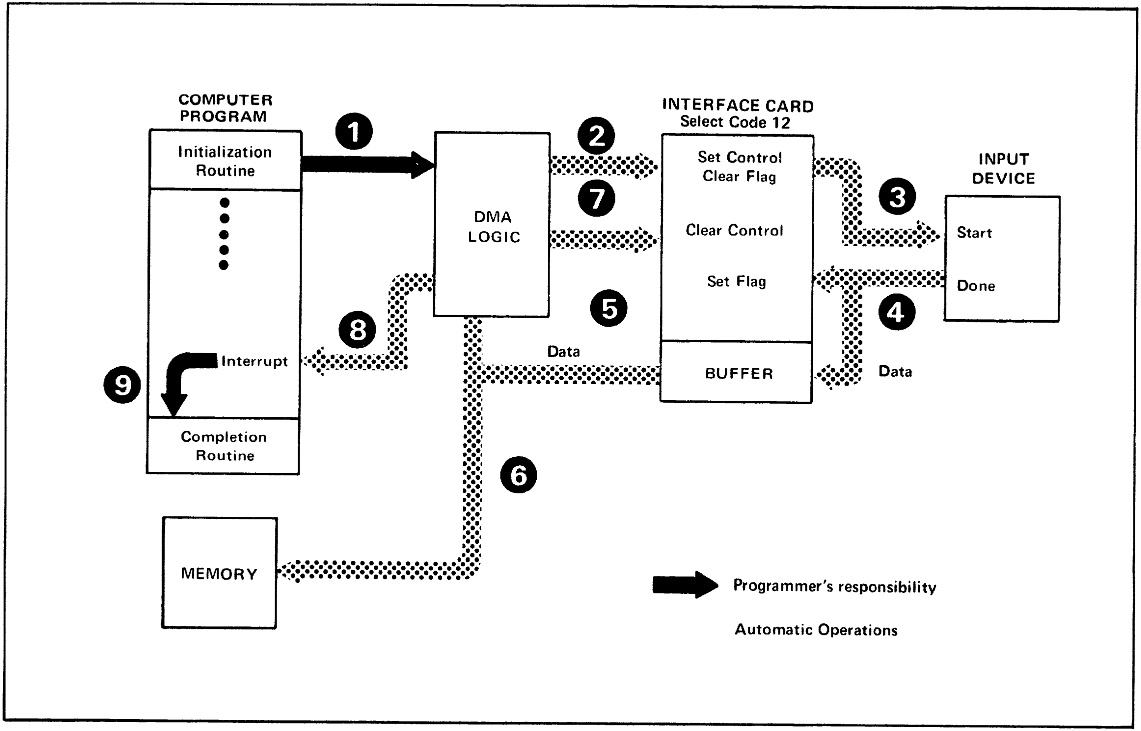


Figure 4.6. DMA Transfers

priority, immediately requests DMA to steal a memory cycle (5) and a word is transferred into (or out of) memory (6). The process now repeats back to the beginning of this paragraph to transfer the next word.

After the specified number of words has been transferred, the control bit is cleared (7). Then DMA generates an interrupt (8), and program control is forced to a completion routine (9), the contents of which is the programmer's responsibility.

4.5.2 DMA INITIALIZATION

The information required to initialize DMA (direction, memory allocation, I/O channel assingment, and block length) are given by three control words. These three words must be addressed specifically to the DMA card. Figure 4.7 shows the format of the three control words.

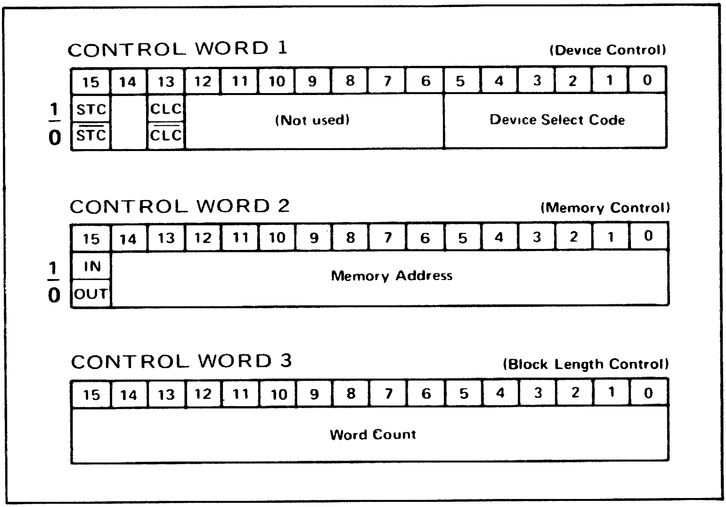


Figure 4.7. DMA Control Word Formats

Control Word 1 (CW1) identifies the I/O channel to be used, and provides for two options, selectable by the programmer as follows:

Bit 15

1:	give STC (in addition to CLF) to I/O channel at end of each DMA cycle (except on last cycle, if input)
0:	no STC

Bit 13

1:	give CLC to I/O channel at end of block transfer
0:	no CLC

Control Word 2 (CW2) gives the starting memory address for the block transfer and Bit 15 determines whether data is to go into memory (1) or out of memory (0).

Control Word 3 (CW3) is the 2's complement of the number of words to be transferred into or out of memory; i.e., the length of block. This number can be from -1 to -32,768, although it is limited in the practical case by available memory.

Table 4.2 gives the basic program sequence for outputting the control words to DMA. As shown in this table, CLC 2 and STC 2 perform switching functions to prepare the logic for either CW2 or CW3. The device is assumed to be in I/O channel 10, and it is also assumed that its start command is STC 10B,C. The sample values of CW1, CW2, CW3 will read a block of 50 words and store these in locations 200 through 261 (octal). STC 6,C starts the DMA operation. A flag-status method for detecting end-of-transfer is used in this example; an interrupt to location 00006 could be substituted for this test.

The program in table 4.2 could easily be changed to operate on channel 2 by changing select codes 2 to 3, and 6 to 7.

One important difference should be noted when doing a DMA input operation from a disc or drum. Due to the asynchronous nature of disc or drum memories and the design of the interface, the order of starting must be reversed from the order given; i.e., start DMA first, then the disc.

LABEL	OPCODE	OPERAND	COMMENTS
ASGN1	LDA	CW1	Fetches control word 1 (CW1) from memory and loads it in A-register.
	OTA	6	Outputs CW1 to DMA Channel 1.
MAR1	CLC	2	Prepares Memory Address Register to receive control word 2 (CW2).
	LDA	CW2	Fetches CW2 from memory and loads it in A-register.
	OTA	2	Outputs CW2 to DMA Channel 1.
WCR1	STC	2	Prepares Word Count Register to receive control word 3 (CW3).
	LDA	CW3	Fetches CW3 from memory and loads it in A-register.
	OTA	2	Outputs CW3 to DMA Channel 1.
STRT1	STC	10B,C	Start input device.
	STC	6B,C	Activate DMA Channel 1.
	SFS JMP	6 *-1	Wait while data transfer takes place or, if interrupt processing is used, continue program.
.	.	.	
.	.	.	
.	.	.	
	HLT		Halt
CW1	OCT	120010	Assignment for DMA Channel 1 (ASGN1); specifies I/O Channel select code address (10_8), STC after each word is transferred, and CLC after final word is transferred.
CW2	OCT	100200	Memory Address Register control. DMA Channel 1 (MAR1); specifies memory input operation and starting memory address (200_8).
CW3	DEC	-50	Word Count Register control. DMA Channel 1 (WCR1); specifies the 2's complement of the number of character words in the block of data to be transferred (50_{10}).

Table 4.2. Program to Initialize DMA

The front panel of the 2100A Computer is available in two configurations: an operator panel (standard) and a controller panel (optional).

The operator panel provides display and control of the working registers, phase status and fault indicators, and operating controls.

The controller panel may be used in applications where an operator panel is seldom required. The panels are easily interchangeable so that, if desired, installations having more than one 2100A Computer may share an operator panel among several units.

This section describes the functions of the controls and indicators on both versions of the panel, plus basic operating procedures.

OPERATING CONTROLS AND INDICATORS 5

5.1 OPERATOR PANEL

Figure 5.1 illustrates the operator panel and briefly describes the function of each control and indicator. The following paragraphs provide additional explanatory information. Functions are grouped according to the type of operation.

5.1.1 16-BIT REGISTERS

The DISPLAY REGISTER displays the contents of any one of the six 16-bit working registers when in the half mode. (Only the S-register is displayed in the run mode.) An illuminated bit pushbutton is a "1"; a non-illuminated bit pushbutton is a "0." The bit content changes state each time the pushbutton is pressed, and the entire display may be cleared by pressing CLEAR DISPLAY.

When power is initially turned on, the S-register is automatically selected. Thereafter, while in half mode, any of the six registers may be selected by pressing the appropriate select switch: A, B, P, M, S, or MEMORY DATA. The register currently selected for display is indicated by lighting of the pushbutton.

After a programmed or manual halt, MEMORY DATA is automatically selected. This causes the contents of the last accessed memory cell to be displayed—which will be the halt instruction code in the case of programmed halts.

As long as a register is being displayed, the original contents of that register may be redisplayed, if altered by pressing DISPLAY REGISTER pushbuttons, simply by pressing the same select pushbutton again (A, B, P, M. S. or MEMORY DATA). However, when any other select pushbutton is pressed (or if the computer is run or stepped) the last indicated display becomes the new contents for that register, and the old contents is lost.

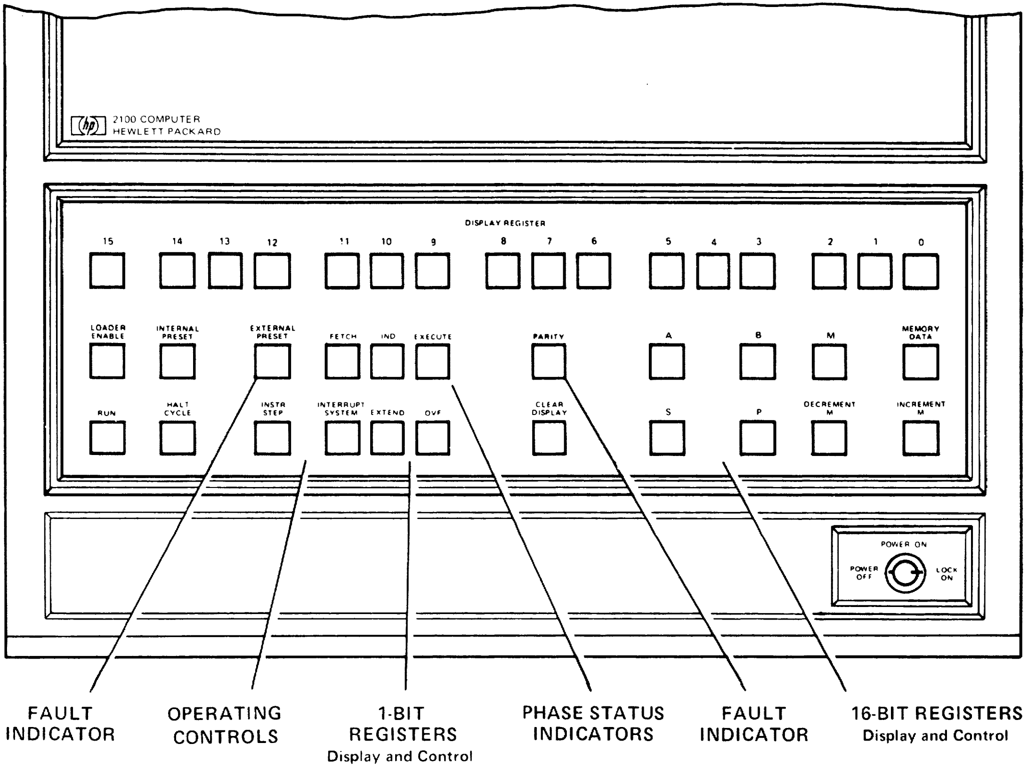


Figure 5.1. Operator Panel Controls and Indicators (Part 1)

16-BIT REGISTERS

DISPLAY REGISTER. Bit light on = 1, off = 0. Press switch to complement any bit.

MEMORY DATA. Press to display contents of location referenced by M. Lit when selected. Can press again to redisplay unmodified contents. selected when computer is halted.

INCREMENT M. Press to increment M. If memory data selected, display is updated.

M. Press to display M-register. Can press again to redisplay unmodified contents.

P. Press to display P-register and set Fetch phase. Can press again to redisplay unmodified contents.

B. Press to display B-register. Can press again to redisplay unmodified contents.

A. Press to display A-register. Can press again to redisplay unmodified contents.

S. Press to display S-register. Can press again to redisplay unmodified contents. Automatically selected in run mode.

CLEAR DISPLAY. Press to clear display register.

1-BIT REGISTERS

OVF. Overflow register. Light on = 1, off = 0. Press to complement.

EXTEND. Extend register. Light on = 1, off = 0. Press to complement.

OPERATING CONTROLS

INTERRUPT SYSTEM. Light on indicates interrupt system enabled. Press to complement.

INSTR STEP. Press to execute single instruction.

EXTERNAL PRESET. Press to clear I/O channels.

INTERNAL PRESET. Set Fetch phase, clear parity error indication and overflow, disable interrupt system and memory protect.

HALT/CYCLE. Halt computer or perform one instruction phase.

LOADER ENABLE. Press to enable/disable loader.

RUN. Start execution, disable panel.

POWER OFF/POWER ON/LOCK ON. Key-operated power switch. Panel disabled in LOCK ON position.

PHASE STATUS INDICATORS

FETCH. Indicates Fetch phase is next.

IND. Indicates Indirect phase is next.

EXECUTE. Indicates Execute phase is next.

FAULT INDICATORS

PARITY. Light on indicates that a memory parity error has occurred (if P.E. HALT mode selected).

EXTERNAL PRESET. Light on indicates a power failure occurred (if location 04 contains HLT).

Note that pressing the M pushbutton displays the address of a memory location, and pressing MEMORY DATA displays the contents of that location. Depending on which of these is selected, consecutive addresses or consecutive contents for adjacent memory cells (either higher or lower) may be displayed by repetitively pressing INCREMENT M or DECREMENT M. These two pushbuttons are only momentarily illuminated when pressed. Pressing the P pushbutton also sets the fetch phase, so that execution may begin (at the location indicated by the P-register) simply by pressing RUN.

5.1.2 FAULT INDICATORS

Provision is made to indicate two possible hardware faults. One is a parity error as a result of reading from memory. If the PARITY light is on, a parity error has occurred. In the halt mode, the light may be turned off by pressing INTERNAL PRESET. In the run mode, the light is turned off by a parity error interrupt, and thus is not ordinarily on long enough to be visible.

The other indicated hardware fault is power failure. If the ARS/ARS switch is set to ARS (auto-restart) and location 04 contains a HLT instruction, the EXTERNAL PRE-SET pushbutton will light on restoration of power, and the machine will halt. The light is turned off by pressing the EXTERNAL PRESET pushbutton. (In a restart routine the light would be turned off by the CLC 04 instruction.)

5.1.3 PHASE STATUS INDICATORS

There are three indicators which signal the state of the computer: FETCH, IND (for indirect), and EXECUTE. The next phase to occur if the computer is run or stepped is the phase indicated by the lighted status indicator. Thus if the FETCH light is on, the computer will fetch an instruction from the address currently pointed to by the P-register when the computer is run or stepped. (It should be noted that indirect references for the extended arithmetic instructions are obtained in an Execute phase, not an Indirect phase.) The indicators are also operative in the run mode.

5.1.4 1-BIT REGISTERS

The contents of the Extend and Overflow registers are continuously displayed by the EXTEND and OVF pushbutton lights (in both halt and run modes). If the pushbutton light is on, the register contents is a "1"; if not on, the register contents is a "0." In the halt mode, the content changes state each time the pushbutton is pressed.

5.1.5 OPERATING CONTROLS

The eight pushbuttons grouped together as operating controls generally control start/stop and other related functions. Since the effects of each pushbutton differ one from another, they are discussed separately below.

INTERRUPT SYSTEM. This pushbutton indicates and controls the state of the interrupt system. When the pushbutton light is on, the interrupt system is enabled (flag set). When the light is off, the interrupt system is disabled (flag clear). Each time the pushbutton is pressed, while the computer is halted, the flag changes state.

INSTR STEP. This pushbutton is used to advance program execution by instruction. The program advances one instruction each time the pushbutton is pressed. If the RUN light stays on, an infinite indirect loop is indicated; press HALT to terminate the loop.

EXTERNAL PRESET. This pushbutton disables the input/output channels. From I/O address 06 and up, all Control flip-flops are cleared and flag flip-flops are set. If the EXTERNAL PRESET pushbutton lights, a power failure has occurred (see description under Fault indicators).

INTERNAL PRESET. This pushbutton presets the computer to the fetch phase, clears the PARITY indicator, clears overflow, and disables both the interrupt system and the memory protect logic.

HALT/CYCLE. In the run mode, this pushbutton is used to halt the computer at the end of the current phase. The pushbutton lights when the computer halts, and all other panel controls become enabled. In the halt mode, the pushbutton may be used to advance program execution by phase. One phase occurs (and the light goes off momentarily) each time the pushbutton is pressed.

LOADER ENABLE. This pushbutton enables access to the basic binary loader (last 64 locations of memory) for the purpose of loading binary programs. When the push button is pressed the light goes on, and stays on as long as the loader is enabled. After a programmed or manual halt, the light goes off and the loader is again disabled. (The loader can also be disabled by pressing the pushbutton again.)

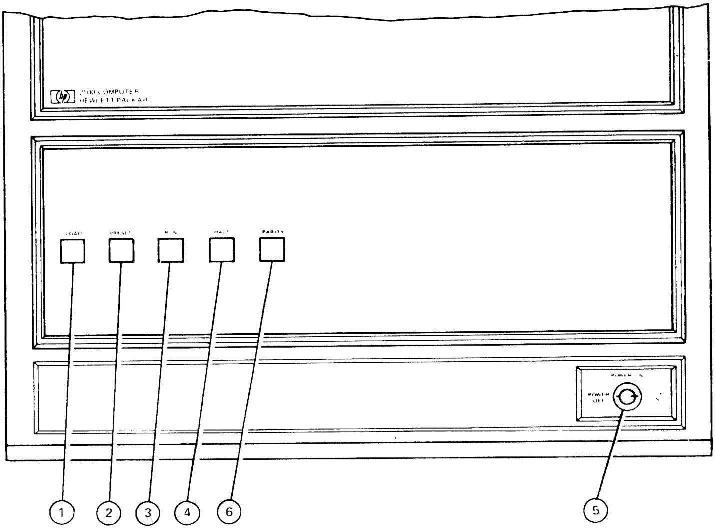
RUN. Pressing RUN starts the computer in the current state. The RUN pushbutton light is on while the computer is in the run mode, and all panel controls are disabled except HALT/CYCLE, DISPLAY REGISTER, and CLEAR DISPLAY. Pressing RUN automatically causes the S-register contents to be displayed, and no other register may be selected while the computer is in the run mode. Thus, to the operator, the DISPLAY REGISTER effectively becomes the S-register. This register may be addressed as select code 01 by programmed instructions, and may be manually altered by the operator.

POWER OFF/POWER ON/LOCK ON. This is a three-position, key-operated switch controlling primary power to the computer. The key is removable only in the horizontal POWER OFF and LOCK ON positions. In the LOCK ON position the panel controls are enabled and the key may not be removed.

If it is desired to inhibit the operation of the automatic restart logic when turning power on, the EXTERNAL PRESET pushbutton may be held depressed while turning the power switch.

5.2 CONTROLLER PANEL

Figure 5.2 illustrates the optional controller panel and briefly describes the function of each control and indicator. The following paragraphs provide additional explanatory information.



OPERATING CONTROLS

1. **LOAD.** After preset, press to load program. Light on during load.
2. **PRESET.** Press to set Fetch phase, turn off I/O channels, interrupt system, memory protect, and indications for parity error and power fail. Also clears A-, B-, and P-registers.
3. **RUN.** Press to start program execution. Light on in run mode.
4. **HALT.** Press to halt execution at end of current phase. Light on when halted.
5. **POWER OFF/POWER ON/LOCK ON.** Key-operated power switch. Panel disabled in LOCK ON position.

FAULT INDICATORS

2. **PRESET.** Light on indicates power failure occurred. (Refer to text.)
6. **PARITY.** Light on indicates a memory parity error has occurred, with P.E. INT/HALT switch set to HALT.

Figure 5.2. Controller Panel Controls and Indicators

PARITY. If the PARITY light is on, a parity error has occurred as a result of reading from memory. In the halt mode, the light may be turned off by pressing the PRESET pushbutton. In the run mode, the light is turned off by a parity error interrupt.

RUN. Pressing RUN starts the computer in the current state. The RUN pushbutton light is on while the computer is in the run mode, and the PRESET pushbutton is disabled.

HALT. This pushbutton is used to halt the computer at the end of the current phase. The pushbutton lights when the computer halts, and the PRESET pushbutton becomes enabled.

PRESET. This pushbutton disables the input/output channels clears Control flip-flops and sets flag flip-flops from I/O address 06 and up) turns off the interrupt system, clears the Overflow, A-, B-, and P-registers, clears the PARITY indicator, disables the memory protect logic, and presets the computer to the fetch phase. Pressing the PRESET pushbutton also clears a power failure indication (PRESET pushbutton light on) if power has failed and is restored. Note that PRESET will light only if the internal ARS/ARS switch is set to ARS and location 04 contains a HLT instruction.

If the RUN pushbutton is pressed after PRESET, the computer will begin program execution from location 0 (P-register = 0). The first two instructions executed will be NOP's (A- and B-registers = 0), and the computer will then begin executing at location 00002. This provides a convenient cold-start linkage in the absence of an operator panel.

LOAD. This pushbutton is used to load a program from a tape reader or disc. In the half mode, pressing the LOAD pushbutton causes the loader starting address to be loaded into the P-register, enables the loader locations, and starts the run mode. The pushbutton light remains on until a programmed or manual halt occurs. The halt disables the loader and turns off the light.

POWER OFF/POWER ON/LOCK ON. The power control switch is not replaced when panels are interchanged. Refer to the description given previously.

5.3 INTERNAL SWITCHES

Although most of the internal switches are intended for checkout or maintenance purposes, two of these are of interest to the user. The following paragraphs describe the functions of these switches. Access to the switches is obtained by removing the computer top cover; each switch is mounted near the top edge of a printed-circuit card, the location of which is specified in the following text.

ARS/$\overline{\text{ARS}}$

The ARS/$\overline{\text{ARS}}$ switch is used to specify the action which the computer should take on recovery from a power failure. With the switch in the ARS position, the computer will interrupt to location 00004 when power returns to normal operating levels; this permits entry to a restart program. With the switch in the ARS position, the computer will halt on recovery of power. The ARS/ARS switch is located on the I/O control card in slot 7.

INT/HALT

The P.E. INT/HALT switch is used to specify the action which the computer should take on detection of a memory parity error. With the switch in the INT position, the computer will interrupt to location 00005 for entry to a parity error subroutine. With the switch in the HALT position, the computer will halt. The P.E. INT/HALT switch is located on the I/O buffer card in slot 8.

5.4 PANEL OPERATION

The following procedures describe, in general, the basic load and run operations for the 2100A Computer. Depending on whether or not a disc is present in the system, loading is accomplished by means of the basic binary loader (BBL) or basic binary disc loader (BBDL). All procedures require that the power-switch key be in the vertical POWER ON position (panel enabled).

5.4.1 LOADING WITH BASIC BINARY LOADER

It is assumed that the basic binary loader program is present in memory, and is properly configured for the channel number of the input device and for the size of memory. Refer to the software operating manual for the procedure required to configure the loader. Loading is accomplished as follows:

a. Turn on the input device and prepare for reading (e.g, load tape in tape reader). The input program must be in binary form, containing absolute addresses.

b. Press S to select the S-register. This will cause the S-register contents to be displayed in the DISPLAY REGISTER.

c. Clear bits 0 and 15 of the display. (These bits are to be set only for certain nonloading check operations; refer to software operating manual.) The status of the remaining bits is not significant.

d. Press P to select the P-register. This will cause the P-register contents to be displayed in the DISPLAY REGISTER.

e. Set the display to the starting address of the basic binary loader, according to table 5-1.

f. Press EXTERNAL PRESET and INTERNAL PRESET. This initializes the external hardware (I/O channels) and the internal hardware (central processor).

g. Press LOADER ENABLE, and then press RUN. The lights for both switches will remain on while the input operation is in progress.

h. When the input device stops, the HALT light will go on, RUN and LOADER ENABLE lights will go off, and the DISPLAY REGISTER should indicate 102077 (octal), with MEMORY DATA automatically selected. The load is complete.

If the halt code is not 102077 when the device stops, there has been an error in the loading process. Two possible error conditions are indicated by the loader, which changes the halt code to identify the type of error. A halt code of 102055 indicates an address error; check if the proper tape is being read, or if it is in backwards. A halt code of 102011 indicates a checksum error; check for possible bad tape, or dirty tape reader or tape.

MEMORY SIZE	STARTING ADDRESS OF LOADER	
	For Paper Tape	For Disc
4K	07700	
8K	17700	17760
12K	27700	27760
16K	37700	37760
24K	57700	57760
32K	77700	77760

Table 5.1. Loader Starting Addresses

5.4.2 LOADING WITH DISC LOADER

If a disc is present in the system, the basic binary disc loader (rather than the basic binary loader) occupies the protected loader locations. This loader allows loading from either disc or paper tape. The choice is made by selecting one of two possible starting addresses, as indicated in table 5.1. For paper tapes the procedure is the same as described above for the basic binary loader; steps "b" and "c" can be omitted.

The following procedures for disc loading assume that the basic binary disc loader is present in memory, and is properly configured for the I/O channel numbers being used and for the size of memory. The input program on disc must be in binary form, containing absolute addresses.

a. Press P to select the P-register. This will cause the P-register contents to be displayed in the DISPLAY REGISTER.

b. Set the display to the starting address in the loader which is appropriate to the input source (disc) and memory size, as indicated in table 5.1.

c. Press EXTERNAL PRESET and INTERNAL PRESET. This initializes the external hardware (I/O channels) and the internal hardware (central processor).

d. Press LOADER ENABLE, and then press RUN.

In the case of disc loading, the load may occur too quickly to detect visually from the panel lights. However, a correct load is indicated (for either tape or disc) by a display of 102077 (octal), with MEMORY DATA automatically selected. (The P-register contents could also be checked. With tape loading, the address should have changed from the first to the last address, plus one, of the loader. With disc loading, the P-register should contain octal 10.)

If the displayed halt code is not 102077 when the load is complete, there has been an error. For disc loading, the error indications are undefinable. For paper tape loading, the loader will alter the halt code to identify the type of error, as described above for basic binary loader operation.

5.4.3 MANUAL LOADING

Short programs may also be loaded manually from the front panel.

a. Press M to select the M-register. This will cause the M-register contents to be displayed in the DISPLAY REGISTER.

b. Set the display to indicate the desired starting address for the program.

c. Press MEMORY DATA. This will cause the current contents of the memory location to be displayed in the DISPLAY REGISTER.

d. Change the displayed contents to the binary instruction code for the first instruction of the program to be loaded. (It may be faster to press CLEAR DISPLAY and begin coding from an all-zero display.)

e. Press INCREMENT M. The contents of the next memory location will be displayed, and the M-register, although not displayed, will be incremented.

f. Enter the next instruction into the DISPLAY REGISTER.

g. Repeat steps "e" and "f" until the entire program has been loaded. To check which location is being displayed, M can be pressed at any time in the procedure to display the current address.

5.4.4 RUNNING PROGRAMS

To run a program after it has been loaded:

a. Press P to select the P-register.

b. Set the display to the starting address of the program.

c. Press EXTERNAL PRESET and INTERNAL PRESET.

d. Press RUN.

The RUN light will be on as long as the program is running. All panel controls except HALT/CYCLE, DISPLAY REGISTER, and CLEAR DISPLAY are disabled. The S-register is automatically selected, and may be manually changed via the DISPLAY REGISTER.

Additionally, if desired, the display and halt controls may also be disabled by turning the power-switch key to the horizontal LOCK ON position. The key may be removed in this position, and thus protect the state of the computer from accidental tampering.

5.5 OPERATION

5.5.1 LOADING PROGRAMS

It is assumed that the loader program is present in memory, and that the loader and the panel are properly configured from the type of loader (paper tape or disc), the channel number of the input device, and the applicable memory size. Refer to the 2100A Installation and Maintenance manual for the procedure required to configure the panel, and to the software operating manual for procedure required to configure the loader. Loading is accomplished as follows:

a. Turn on the input device and prepare for reading (e.g., load tape in tape reader). The input program must be in binary form, containing absolute addresses.

b. Press PRESET. This initializes both the external hardware (I/O channels) and the internal hardware (central processor).

c. Press LOAD. The LOAD light will be on and will remain on during the load (or until the pushbutton is released in the case of disc loading). No error checking is provided.

5.5.2 RUNNING PROGRAMS

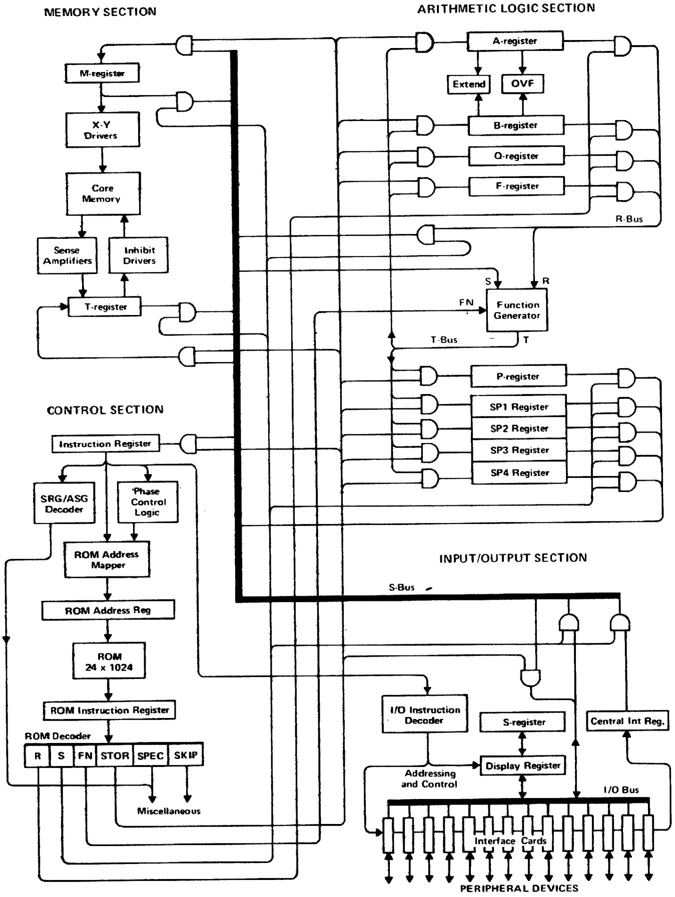
To run the loaded program, press PRESET and then press RUN.

The PRESET switch causes the A-, B-, and P-registers to be cleared, thus causing execution to begin at location 00000 (A-register). The computer executes the NOP instruction contained in the A-register (all-zero word), and also the NOP in the B-register. Then, in location 00002, a JMP instruction causes a jump to the starting instruction of the program.

The RUN light will be on as long as the program is running. Only the HALT switch is enabled. However, even this switch may be disabled by turning the power-switch key to the horizontal LOCK ON position. The key may be removed in this position, and thus protect the state of the computer from accidental tampering.

FUNCTIONAL BLOCK DIAGRAM A

FUNCTIONAL BLOCK DIAGRAM



PROCESSOR LOGIC ELEMENTS B

MEMORY SECTION

M-Register. Contains binary address of memory cell being accessed. Contents gated to or from S-bus by S and STOR fields (respectively) of ROM instruction word.

X-Y Drivers. Current drivers which strobe all 17 cores in a given memory location, one direction for reading, the opposite direction for writing.

Core Memory. Array of magnetic cores for data storage. Magnetization direction of each core indicates "1" bit or "0" bit. (17 bits per location.)

Sense Amplifiers. Pulse amplifiers to detect which of the 17 cores change state when reading the contents of one location. Resulting signals cause duplication of the bit pattern into the T-register. (17th bit goes to parity checking logic, not shown.)

Inhibit Drivers. Current drivers which prevent certain cores from changing state (according to the bit pattern in the T-register) when writing into memory. Causes duplication of T-register contents into the memory location.

T-register. 16-bit register to receive data from memory, and hold data for storage into memory. Contents can be gated to or from S-bus by ROM S or STOR fields.

CONTROL SECTION

Instruction Register. 16-bit register to receive instruction word from T-register in fetch phase. Loaded from S-bus by ROM STOR field.

SRG/ASG Decoder. Register reference instructions are partially decoded separate from ROM. Resulting control signals directly affect A-, B-, or P-registers.

Phase Control Logic. Causes ROM address mapper to set up a ROM address corresponding to the current instruction phase.

ROM Address Mapper. Uses the instruction register code to find the ROM starting address for an instruction, and the address for each phase of that instruction.

ROM Address Register. Contains the binary address of the ROM location being read out.

ROM. (Read-only memory.) A matrix of permanently stored instruction codes, addressable to read out any stored code on command. (24 bits per location.)

ROM Instruction Register. 24-bit register to receive ROM instruction words.

ROM Decoder. Decodes ROM instruction codes into control signals, to select which register to read onto the R- and S-buses, as well as where to store S-bus data; also numerous other functions.

ARITHMETIC LOGIC SECTION

A-register. 16-bit accumulator. Loaded from T-bus by ROM STOR field, read to R-bus by R field.

B-register. Second accumulator, same as A-register.

Extend. One-bit register used to extend the A- or B- register to 17 bits. Can also be used independently.

OVF. (Overflow.) One-bit register used to signify an arithmetic overflow due to arithmetic operations with the A- or B-registers. Can also be used independently.

Q-register. 16-bit left-shifting register, used to accumulate quotient in arithmetic division. Not externally accessible.

F-register. Same as Q-register, except accumulates division remainder.

R-bus. 16-bit data bus, one of two data inputs to the function generator. ROM R field reads 1 of 4 registers onto this bus. Can be gated to S-bus by S field.

Function Generator. Performs a specified function (FN) on one or both of the R- and S-bus inputs, and puts the result onto the T-bus. Functions include: addition, subtraction, boolean operations, increment, decrement, etc.

T-Bus. 16-bit data bus to transfer data modified by the function generator to any of nine registers.

P-register. 16-bit register used to hold the address of the current program instruction.

SP(1-4) Registers. 16-bit temporary storage registers used by ROM only.

INPUT/OUTPUT SECTION

I/O Instruction Decoder. Input/output instructions are partially decoded separate from ROM. Resulting signals provide addressing and control functions to the I/O system.

S-register. 16-bit data register. Can be loaded via display register in halt mode. In run mode, S-register is locked to display register; is addressable by select code 01.

Display Register. In halt mode, provides manual loading facility for other registers. In run mode, may be gated via I/O bus to or from S-bus, using select code 01 with S and STOR fields.

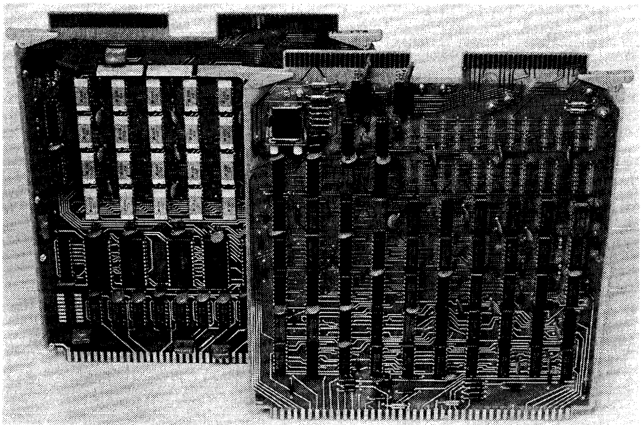
Central Interrupt Register. Six-bit register, holds the address of the most recently interrupting function or device.

I/O Bus. 16-bit data bus accessible to all I/O interface cards. Can be gated to or from the S-bus by ROM S and STOR fields.

Interface Cards. One card per I/O channel, allows direct cable connection of peripheral devices to the input/output section of the computer.

Assembler Reference Manual

Writeable Control Store (left) is used to develop microprograms that extend the 2100A Computer's instruction set. Microprograms are then committed to read-only memories and installed on the microprocessor board (right).



CONTENTS

INTRODUCTION

The Assembler and the Extended Assembler translate symbolic source language instructions into an object program for execution on the computer. The source language provides mnemonic machine operation codes, assembler directing pseudo codes, and symbolic addressing. The assembled program may be absolute or relocatable.

The source program may be assembled as a complete entity or it may be subdivided into several subprograms (or a main program and several subroutines), each of which may be assembled separately. The relocating loader loads the program and Links the subprograms as required. The Basic Binary Loader or Basic Binary Disc Loader loads absolute programs.

Input for the Assembler is prepared on paper tape or cards; the Assembler punches the binary program on paper tape in a format acceptable to the loader.

GENERAL DESCRIPTION 1

1.1 ASSEMBLY PROCESSING

The Assembler is a two pass system, or, if both punch and list output are requested, a three pass system on a minimum configuration. A pass is defined as a processing cycle of the source program input.

In the first pass, the Assembler creates a symbol table from the names used in the source statements. It also checks for certain possible error conditions and generates diagnostic messages if necessary.

During pass two, the Assembler again examines each statement in the source program along with the symbol table and produces the binary program and a program listing. Additional diagnostic messages may also be produced.

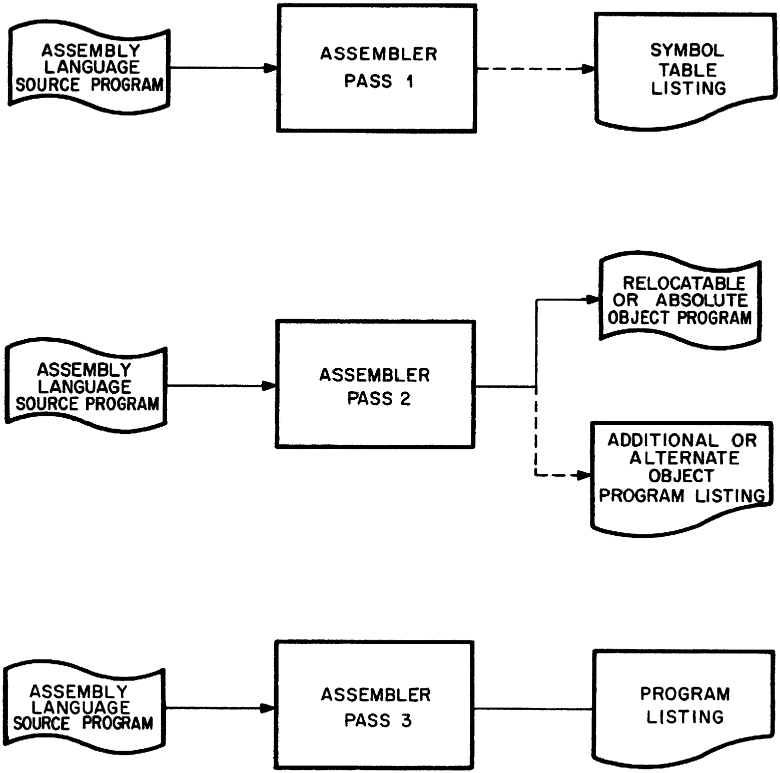
If only one output device is available and if both the binary output and the list output are requested, the listing function is deferred and performed as pass three.

When using the Assembler with a mass storage device the source program is written on the device during the first pass; the second pass of the source is read from the mass storage.

1.2 SYMBOLIC ADDRESSING

Symbols may be used for referring to machine instructions, data, constants, and certain other pseudo operations. A symbol represents the address for a computer word in memory. A symbol is defined when it is used as a label for a location in the program, a name of a common storage segment, the label of a data storage area or constant, the label of an absolute or relocatable value, or a location external to the program.

Through use of simple arithmetic operators, symbols may be combined with other symbols or numbers to form an expression which may identify a location other than that specifically named by a symbol. Symbols appearing in operand expressions, but not specifically defined, and symbols that are defined more than once are considered to be in error by the Assembler.


ASSEMBLY LANGUAGE SOURCE PROGRAM
ASSEMBLER PASS 1
SYMBOL TABLE LISTING
ASSEMBLY LANGUAGE SOURCE PROGRAM
ASSEMBLER PASS 2
RELOCATABLE OR ABSOLUTE OBJECT PROGRAM
ADDITIONAL OR ALTERNATE OBJECT PROGRAM LISTING
ASSEMBLY LANGUAGE SOURCE PROGRAM
ASSEMBLER PASS 3
PROGRAM LISTING

HP ASSEMBLER PROCESSING

1.3 PROGRAM RELOCATION

Relocatable programs may be relocated in core by the relocating loader; the location of the program origin and all subsequent instructions is determined at the time the program is loaded.

A relocatable program is assembled assuming a starting location of zero. All other instructions and data areas are assembled relative to this zero base. When the program is loaded, the relocatable operands are adjusted to correspond with the actual locations assigned by the loader.

The starting locations of the common storage area and the base page portion of the program are always established by the loader. References to the common area are common relocatable. References to the base page portion of the program are base page relocatable. If a program refers to the common area or makes use of the base page via the ORB pseudo instruction, the program must also be relocatable.

If a program is to be relocatable, all subprograms comprising the program must be relocatable; all memory reference operands must be relocatable expressions or literals, or have an absolute value of less than 100_8.

1.4 PROGRAM LOCATION COUNTERS

The Assembler maintains a counter, called the program location counter, that assigns consecutive memory addresses to source statements.

The initial value of the program location counter is established according to the use of either the NAM or ORG pseudo operation at the start of the program. The NAM operation causes the program location counter to be set to zero for a relocatable program; the ORG operation specifies the absolute starting location for an absolute program.

Through use of the ORB pseudo operation a relocatable program may specify that certain operations or data areas be allocated to the base page. If so, a separate counter, called the base page location counter, is used in assigning these locations.

1.5 ASSEMBLY OPTIONS

Parameters specified with the first statement, the control statement, define the output to be produced by the Assembler:†

Absolute – The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity; external symbols, common storage references, and entry points are not permitted.

Relocatable – The program may be located anywhere in memory. All operands which refer to memory locations are adjusted as the program is loaded. Operands, other than those referring to the first 64 locations, must be relocatable expressions. Subprograms may contain external symbols and entry points, and may refer to common storage.

Binary output – An absolute or relocatable program is to be punched on paper tape.

List output – A program listing is produced either during pass two or pass three.

Table print – List the symbol table at the end of the first pass.

Selective assembly - Sections of the program may be included or excluded at assembly time depending on the option used.

† See Chapter 5 for complete details.

INSTRUCTION FORMAT 2

A source language statement consists of a label, an operation code, an operand, and comments. The label is used when needed as a reference by other statements. The operation code may be a mnemonic machine operation or an assembly directing pseudo code. An operand may be an expression consisting of an alphanumeric symbol, a number, a special character, or any of these combined by arithmetic operations. (For the Extended Assembler, an operand may also be a literal.) Indicators may be appended to the operand to specify certain functions such as indirect addressing. The comments portion of the statement is optional.

2.1 STATEMENT CHARACTERISTICS

The fields of the source statement appear in the following order:

Label

Opcode

Operand

Comments

Field Delimiters

One or more spaces separate the fields of a statement. An end-of-statement mark terminates the entire statement. On paper tape this mark is a return, (CR), and line feed, (LF). † A single space following the end-of-statement mark from the previous source statement is the null field indicator of the label field.

Character Set

The characters that may appear in a statement are as follows:

A through Z

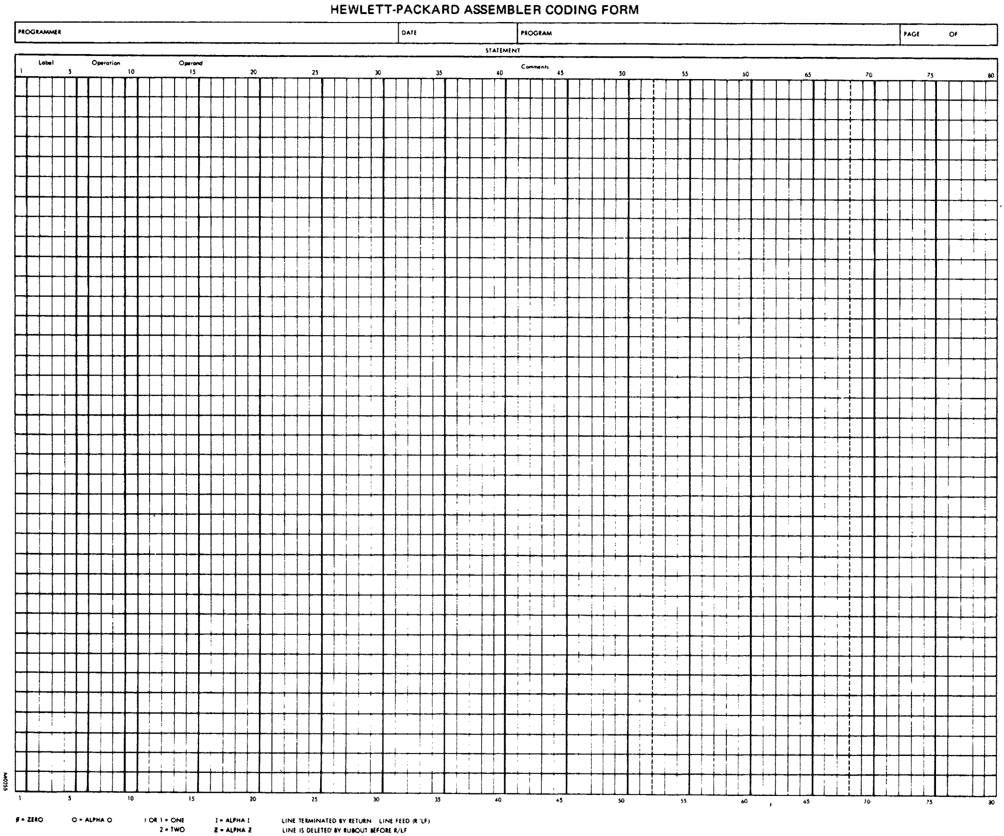
0 through 9

. (period)

* (asterisk)

† A circled symbol (e.g., (CR)) represents an ASCII code or Teleprinter key.

SAMPLE CODING FORM
(Actual Size 11 × 13-1/2)



HEWLETT-PACKARD ASSEMBLER CODING FORM

PROGRAMMER | DATE | PROGRAM | PAGE OF

STATEMENT

Label | Operation | Operand | Comments

1 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80

M0255

Ø = ZERO   O = ALPHA O   I OR 1 = ONE   I = ALPHA I   LINE TERMINATED BY RETURN LINE FEED (R/LF)
2 = TWO   Z = ALPHA Z   LINE IS DELETED BY RUBOUT BEFORE R/LF

+	(plus)
-	(minus)
,	(comma)
=	(equals)
( )	(parentheses)
	(space)

Any other ASCII characters may appear in the Remarks field (See Appendix A).

The letters A through Z, the numbers 0 through 9, and the period may be used in an alphanumeric symbol. In the first position in the Label field, an asterisk indicates a comment; in the Operand field, it represents the value of the program location counter for the current instruction. The plus and minus are used as operators in arithmetic address expressions. The comma separates several operation codes, or an expression and an indicator in the Operand field. An equals sign indicates a literal value. The parentheses are used only in the COM pseudo instruction.

Spaces separate fields of a statement. They may also be used to establish the format of the output list. Within a field they may be used freely when following +, -, , , or (.

Statement Length

A statement may contain up to 80 characters including blanks, but excluding the end-of-statement mark. Fields beginning in characters 73 - 80 are not processed by the Assembler.

2.2 LABEL FIELD

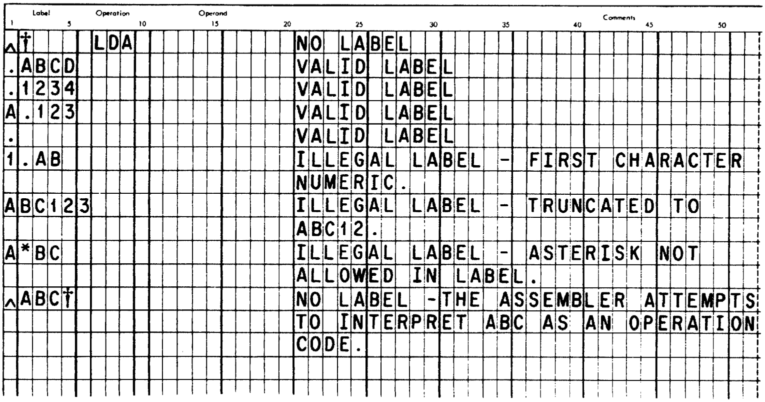
The Label field identifies the statement and may be used as a reference point by other statements in the program.

The field starts in position one of the statement; the first position following an end-of-statement mark for the preceding statement. It is terminated by a space. A space in position one is the null field indicator for the label field; the statement is unlabeled.

Label Symbol

A label must be symbolic. It may have one to five characters consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period. A label of more than five characters could be entered on the source language tape, but the Assembler flags this condition as an error and truncates the label from the right to five characters.

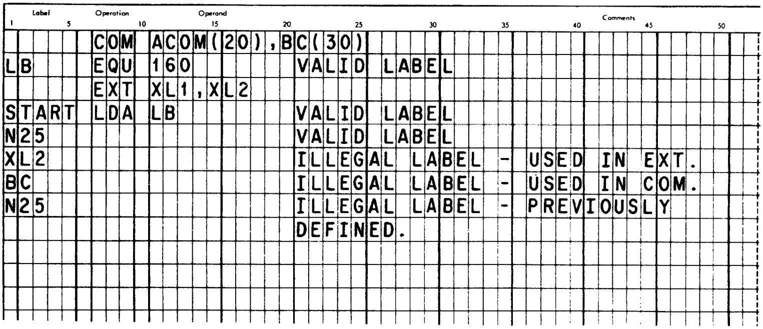
Examples:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
^†    LDA           NO LABEL
.ABCD               VALID LABEL
.1234               VALID LABEL
A.123               VALID LABEL
.                   VALID LABEL
1.AB                ILLEGAL LABEL - FIRST CHARACTER
                    NUMERIC.
ABC123              ILLEGAL LABEL - TRUNCATED TO
                    ABC12.
A*BC                ILLEGAL LABEL - ASTERISK NOT
                    ALLOWED IN LABEL.
^ABC†               NO LABEL -THE ASSEMBLER ATTEMPTS
                    TO INTERPRET ABC AS AN OPERATION
                    CODE.
```

Each label must be unique within the program; two or more statements may not have the same symbolic name. Names which appear in the Operand field of an EXT or COM pseudo instruction may not also be used as statement labels in the same subprogram.

Examples:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      COM ACOM(20),BC(30)
LB    EQU 160       VALID LABEL
      EXT XL1,XL2
START LDA LB        VALID LABEL
N25                 VALID LABEL
XL2                 ILLEGAL LABEL - USED IN EXT.
BC                  ILLEGAL LABEL - USED IN COM.
N25                 ILLEGAL LABEL - PREVIOUSLY
                    DEFINED.
```

† The caret symbol, ^, indicates the presence of a space.

Asterisk

An asterisk in position one indicates that the entire statement is a comment. Positions 2 through 80 are available; however, positions 1 through 68 only are printed as part of the assembly listing on the 2752A Teleprinter. An asterisk within the Label field is illegal in any position other than one.

2.3 OPCODE FIELD

The operation code defines an operation to be performed by the computer or the Assembler. The Opcode field follows the Label field and is separated from it by at least one space. If there is no label, the operation code may begin anywhere after position one. The Opcode field is terminated by a space immediately following an operation code. Operation codes are organized in the following categories:

- Machine operation codes
  - Memory Reference
  - Register Reference
  - Input/Output, Overflow, and Halt
  - Extended Arithmetic Unit
- Pseudo operation codes
  - Assembler control
  - Object program linkage
  - Address and symbol definition
  - Constant definition
  - Storage allocation
  - Arithmetic subroutine calls
  - Assembly Listing Control (Extended Assembler)

Operation codes are discussed in detail in Chapters 3 and 4.

2.4 OPERAND FIELD

The meaning and format of the Operand field depend on the type of operation code used in the source statement. The field follows the Opcode field and is separated from it by at least one space. It is terminated by a space except when the space follows , + - (or, if there are no comments, by an end-of-statement mark.

The Operand field may contain an expression consisting of one of the following:

Single symbolic term

Single numeric term

Asterisk

Combination of symbolic terms, numeric terms, and the asterisk joined by the arithmetic operators + and -.

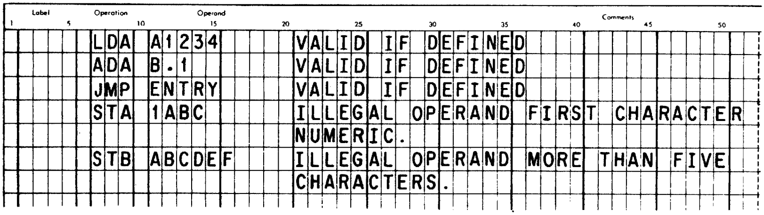
An expression may be followed by a comma and an indicator.

Programs being assembled by the Extended Assembler may also contain a literal value in the Operand field.

Symbolic Terms

A symbolic term may be one to five characters consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period.

Examples:



Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      LDA A1234     VALID IF DEFINED
      ADA B.1       VALID IF DEFINED
      JMP ENTRY     VALID IF DEFINED
      STA 1ABC      ILLEGAL OPERAND FIRST CHARACTER
                    NUMERIC.
      STB ABCDEF    ILLEGAL OPERAND MORE THAN FIVE
                    CHARACTERS.
```

A symbol used in the Operand field must be a symbol that is defined elsewhere in the program in one of the following ways:

As a label in the Label field of a machine operation

As a label in the Label field of a BSS, ASC, DEC, OCT, DEF, ABS, EQU or REP pseudo operation

As a name in the Operand field of a COM or EXT pseudo operation

As a label in the Label field of an arithmetic subroutine pseudo operation

The value of a symbol is absolute or relocatable depending on the assembly option selected by the user. The Assembler assigns a value to a symbol as it appears in one of the above fields of a statement. If a program is to be loaded in absolute form, the values assigned by the assembler remain fixed. If the program is to be relocated, the actual value of a symbol is established on loading. A symbol may also be made absolute through use of the EQU pseudo instruction.

A symbolic term may be preceded by a plus or minus sign. If preceded by a plus or no sign, the symbol refers to its associated value. If preceded by a minus sign, the symbol refers to the two's complement of its associated value. A single negative symbolic operand may be used only with the ABS pseudo operation.

Numeric Terms

A numeric term may be decimal or octal. A decimal number is represented by one to five digits within the range 0 to 32767. An octal number is represented by one to six octal digits followed by the letter B; (0 to 177777B).

If a numeric term is preceded by a plus or no sign, the binary equivalent of the number is used in the object code. If preceded by a minus sign, the two's complement of the binary equivalent is used. A negative numeric operand may be used only with the DEX, DEC, OCT, and ABS pseudo operations.

In an absolute program, the maximum value of a numeric operand depends on the type of machine or pseudo instruction. In a relocatable program, the value of a numeric operand may not exceed 77B. Numeric operands are absolute. Their value is not altered by the assembler or the loader.

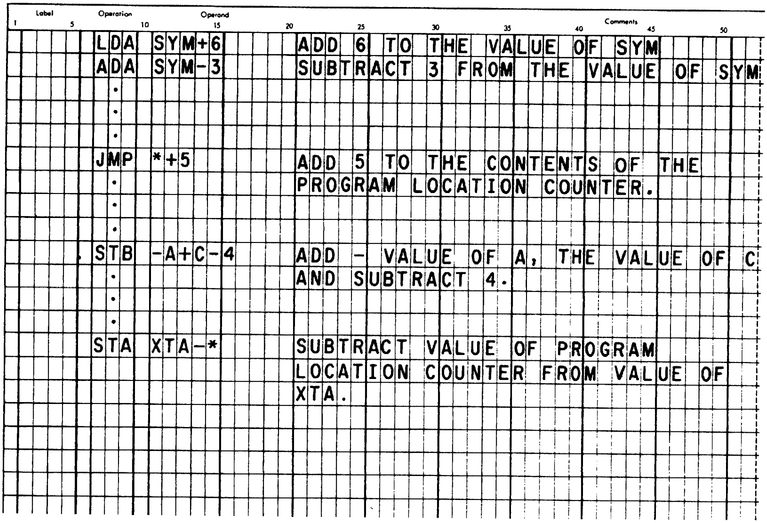
Asterisk

An asterisk in the Operand field refers to the value in the program location counter (or base page location counter) at the time the source program statement is encountered. The asterisk is considered a relocatable term in a relocatable program.

Expression Operators

The asterisk, symbols, and numbers may be joined by the arithmetic operators + and - to form arithmetic address expressions. The Assembler evaluates an expression and produces an absolute or relocatable value in the object code.

Examples:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      LDA SYM+6     ADD 6 TO THE VALUE OF SYM
      ADA SYM-3     SUBTRACT 3 FROM THE VALUE OF SYM
       .
       .
       .
      JMP *+5       ADD 5 TO THE CONTENTS OF THE
       .            PROGRAM LOCATION COUNTER.
       .
       .
      STB -A+C-4    ADD - VALUE OF A, THE VALUE OF C
       .            AND SUBTRACT 4.
       .
       .
      STA XTA-*     SUBTRACT VALUE OF PROGRAM
                    LOCATION COUNTER FROM VALUE OF
                    XTA.
```

Evaluation of Expressions

An expression consisting of a single operand has the value of that operand. An expression consisting of more than one operand is reduced to a single value. In expressions containing more than one operator, evaluation of the expression proceeds from left to right. The algebraic expression A-(B-C+5) must be represented in the Operand field as A-B+C-5. Parentheses are not permitted in operand expressions for the grouping of operands.

The range of values that may result from an operand expression depends on the type of operation. The Assembler evaluates expressions as follows:†

Pseudo Operations	modulo 2^{15}-1
Memory Reference	modulo 2^{10}-1
Input/Output	2^6 - 1 (maximum value)

† The evaluation of expressions by the Assembler is compatible with the addressing capability of the hardware instructions (e.g., up to 32K words through Indirect Addressing). The user must take care not to create addresses which exceed the memory size of the particular configuration.

Expression Terms

The terms of an expression are the numbers and the symbols appearing in it. Decimal and octal integers, and symbols defined as being absolute in an EQU pseudo operation are absolute terms. The asterisk and all symbols that are defined in the program are relocatable or absolute depending on the type of assembly. Symbols that are defined as external may appear only as single term expressions.

Within a relocatable program, terms may be program relocatable, base page relocatable, or common relocatable. A symbol that names an area of common storage is a common relocatable term. A symbol that is allocated to the base page is a base page relocatable term. A symbol that is defined in any other statement is a program relocatable term. Within one expression all relocatable terms must be base page relocatable, program relocatable, or common relocatable; the three types may not be mixed.

Absolute and Relocatable Expressions

An expression is absolute if its value is unaffected by program relocation. An expression is relocatable if its value changes according to the location into which the program is loaded. In an absolute program, all expressions are absolute. In a relocatable program, an expression may be base page relocatable, program relocatable, common relocatable, or absolute (if less than 100_8) depending on the definition of the terms composing it.

Absolute Expressions

An absolute expression may be any arithmetic combination of absolute terms. It may also contain relocatable terms alone, or in combination with absolute terms. If relocatable terms do appear, there must be an even number of them; they must be of the same type; and they must be paired by sign (a negative term for each positive term). The paired terms do not have to be contiguous in the expression. The pairing of terms by type cancels the effect of relocation; the value represented by the pair remains constant.

An absolute expression reduces to a single absolute value. The value of an absolute multiterm expression may be negative only for ABS pseudo operations. A single numeric term also may be negative in an OCT, DEX, or DEC pseudo instruction. In a relocatable program the value of an absolute expression must be less than 100_8 for instructions that reference memory locations (Memory Reference, DEF, Arithmetic subroutine calls).

Examples:

If P_1 and P_2 are program relocatable terms; B_1 and B_2, base page relocatable; C_1 and C_2, common relocatable; and A, an absolute term; then the following are absolute terms:

$A-C_1+C_2$

$A+A$

$*-P_1$

B_1-*

$A-P_1+P_2$

P_1-P_2

B_1-B_2-A

$-P_1+P_2$

C_1-C_2+A

B_1-B_2

$-C_1+C_2+A$

$-A-P_1+P_2$

The asterisk is base page relocatable or program relocatable depending on the location of the instruction.

Relocatable Expressions

A relocatable expression is one whose value is changed by the loader. All relocatable expressions must have a positive value.

A relocatable expression may contain any odd number of relocatable terms, alone, or in combination with absolute terms. All relocatable terms must be of the same type. Terms must be paired by sign with the odd term being positive.

A relocatable expression reduces to a single positive relocatable term, adjusted by the values represented by the absolute terms and paired relocatable terms associated with it.

Examples:

If P_1, P_2, and P_3 are program relocatable terms; B_1, B_2, and B_3, base page relocatable; C_1, C_2 and C_3, common relocatable; and A, an absolute term; then the following are relocatable terms:

P_1-A

$P_1-P_2+P_3$

$*+A$

$A+B_1$

$B_1-B_2+B_3-A$

$*+P_1-*$

C_1-A

$C_1-C_2+C_3$

$*-P_1+P_2$

$A+C_1$

$C_1-C_2+C_3-A$

P_1-P_2+*

B_1+A

C_1+A

$*-A$

$-A-P_1+P_2+P_3$

$A+*$

$-C_1+C_2+C_3$

Literals

Actual literal values may be specified as operands in relocatable programs to be assembled by the Extended Assembler. The Extended Assembler converts the literal to its binary value, assigns an address to it, and substitutes this address as the operand. Locations assigned to literals are those immediately following the last location used by the program.

A literal is specified by using an equal sign and a one-character identifier defining the type of literal. The actual literal value is specified immediately following this identifier; no spaces may intervene.

The identifiers are:

=D	a decimal integer, in the range -32767 to 32767, including zero.†
=F	a floating-point number; any positive or negative real number in the range 10^{-38} to 10^{38}, including zero.†
=B	an octal integer, one to six digits, $b_1b_2b_3b_4b_5b_6$, where b_1 may be 0 or 1, and b_2-b_7 may be 0 to 7.†
=A	two ASCII characters.†
=L	an expression which, when evaluated, will result in an absolute value. All symbols appearing in the expression must be previously defined.

If the same literal is used in more than one instruction, only one value is generated, and all instructions using this literal refer to the same location.

Literals may be specified only in the following memory reference instructions and pseudo instructions:

ADA	ADB	AND	MPY	may use =D, =B, =A, =L
LDA	LDB	XOR	DIV	
CPA	CPB	IOR		

DLD	FAD	may use =F
FMP	FSB	
FDV		

† See CONSTANT DEFINITION, Section 4.4.

Examples:

LDA	=D798Ø	A-Register is loaded with the binary equivalent of 798Ø$_{1Ø}$.
IOR	=B777	Inclusive "or" is performed with contents of A-Register and 777_8.
LDA	=ANO	A-Register is loaded with binary representation of ASCII characters NO.
LDB	=LZETZ-ZOOM+68	B-Register is loaded with the value resulting from the absolute expression.
FMP	=F39.75	Contents of A- and B-Registers multiplied by floating-point constant 39.75.

Indirect Addressing

The HP computers provide an indirect addressing capability for Memory Reference instructions. The operand portion of an indirect instruction contains an address of another location rather than an actual operand. The secondary location may be the operand or it may be indirect also and give yet another location, and so forth. The chaining ceases when a location is encountered that does not contain an indirect address. Indirect addressing provides a simplified method of address modifications as well as allowing access to any location in core.

The Assembler allows specification of indirect addressing by appending a comma and the letter I to any Memory Reference operand other than one referring to an external symbol. The actual operand of the instruction may be given in a DEF pseudo operation; this pseudo operation may also be used to indicate further levels of indirect addressing.

Examples:

```
Label     Operation     Operand                          Comments
1   5     10            15          20       25       30       35       40       45       50
AB    LDA SAM,I     EACH TIME THE ISZ IS EXECUTED,
AC    ADA SAM,I     THE EFFECTIVE OPERAND OF AB AND
AD    ISZ SAM       AC CHANGE ACCORDINGLY.
       .
       .
       .
SAM   DEF ROGER
```

A relocatable assembly language program, however, may be designed without concern for the pages in which it will be stored; indirect addressing is not required in the source language. When the program is being loaded, the loader provides indirect addressing whenever it detects an operand which does not fall in the current page or the base page. The loader substitutes a reference to the base page and then stores an indirect address in this referenced location. References to the same operand from other pages will be linked through the same location in the base page.

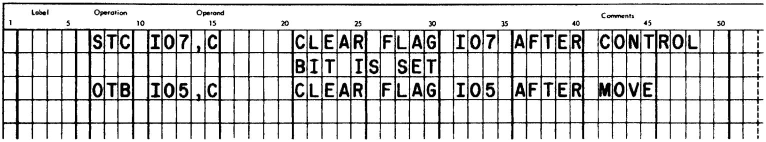
Base Page Addressing

The computer provides a capability which allows the Memory Reference instructions to address either the current page or the base page. The Assembler or the loader adjusts all instructions in which the operands refer to the base page; specific notation defining an operand as a base page reference is not required in the source program.

Clear Flag Indicator

The majority of the input/output instructions can alter the status of the input/output interrupt flag after execution or after the particular test is performed. In source language, this function is selected by appending a comma and a letter C to the Operand field.

Examples:



Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      STC I07,C     CLEAR FLAG I07 AFTER CONTROL
                    BIT IS SET
      OTB I05,C     CLEAR FLAG I05 AFTER MOVE
```

2.5 COMMENTS FIELD

The Comments field allows the user to transcribe notes on the program that will be listed with source language coding on the output produced by the Assembler. The field follows the Operand field and is separated from it by at least one space. The end-of-statement mark, (CR) (LF), or the 80th character in the entire statement terminates the field. If the listing is to be produced on the 2752A Teleprinter, the total statement length, excluding the end-of-statement mark, should not exceed 52 characters, the width of the source language portion of the listing. Statements consisting solely of comments may contain up to 68 characters including the asterisk in the first position. On the list output, statements consisting entirely of comments begin in position 5 rather than 21 as with other source statements.

If there is no operand present, the Comments field should be omitted in the NAM and END pseudo operations and in the input/output statements, SOC, SOS, and HLT instruction. If a comment is used, the Assembler attempts to interpret it as an operand.

MACHINE INSTRUCTIONS 3

The HP Assembler language machine instruction codes take the form of three-letter mnemonics. Each source statement corresponds to a machine operation in the object program produced by the Assembler.

Notation used in representing source language instruction is as follows:

label	Optional statement label
m	Memory location -- an expression
I	Indirect addressing indicator
sc	Select code -- an expression
C	Clear interrupt flag indicator
comments	Optional comments
[ ]	Brackets defining a field or portion of a field that is optional
{ }	Brackets indicating that one of the set may be selected.
lit	literal

3.1 MEMORY REFERENCE

Memory Reference instructions perform arithmetic, logical and jump operations on the contents of the locations in core and the registers. An instruction may directly address the 2048 words of the current and base pages. If required, indirect addressing may be utilized to refer to all 32,768 words of memory. Expressions in the operand field are evaluated modulo 2^{10} .

If the program is to be assembled in relocatable form, the operand field may contain relocatable expressions or absolute expressions which are less than 100_8 in value. If the program is to be absolute, the operands may be any expressions consistent with the location of the program. Literals may not be used in an absolute program. Absolute programs must be complete entities; they may not refer to external subroutines or common storage.

Jump and Increment-Skip

Jump and Increment-Skip instructions may alter the normal sequence of program execution.

label	JMP	m [, I]	comments

Jump to m. Jump indirect inhibits interrupt until the transfer of control is complete.

label	JSB	m [, I]	comments

Jump to subroutine. The address for label+1 is placed into the location represented by m and control transfers to m+1. On completion of the subroutine, control may be returned to the normal sequence by performing a JMP m, I.

label	ISZ	m [, I]	comments

Increment, then skip if zero. ISZ adds 1 to the contents of m. If m then equals zero, the next instruction in memory is bypassed.

Add, Load, and Store

Add, Load, and Store instructions transmit and alter the contents of memory and of the A- and B-Registers. A literal, indicated by ''lit'', may be either =D, =B, =A, or =I type.

label	ADA	{m [, I] lit}	comments

Add the contents of m to A.

label	ADB	{m [, I] lit}	comments

Add the contents of m to B.

label	LDA	{m [, I] lit}	comments

Load A from m.

label	LDB	{m [, I] lit}	comments

Load B from m.

label	STA	m [,I]	comments

Store contents of A in m.

label	STB	m [,I]	comments

Store contents of B in m.

In each instruction, the contents of the sending location is unchanged after execution.

Logical Operations

The Logical instructions allow bit manipulation and the comparison of two computer words.

label	AND	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	comments

The logical product of the contents of m and the contents of A are placed in A.

label	XOR	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	comments

The modulo-two sum (exclusive "or") of the bits in m and the bits in A is placed in A.

label	IOR	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	comments

The logical sum (inclusive "or") of the bits in m and the bits in A is placed in A.

label	CPA	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	comments

Compare the contents of m with the contents of A. If they differ, skip the next instruction; otherwise, continue.

label	CPB	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	comments

Compare the contents of m with the contents of B. If they differ, skip the next instruction; otherwise, continue.

3.2 REGISTER REFERENCE

The Register Reference instructions include a Shift-Rotate group, an Alter-Skip group, and NOP (no-operation). With the exception of NOP, they have the capability of causing several actions to take place during one memory cycle. Multiple operations within a statement are separated by a comma.

Shift-Rotate Group

This group contains 19 basic instructions that can be combined to produce more than 500 different single cycle operations.

CLE	Clear E to zero
ALS	Shift A left one bit, zero to least significant bit. Sign unaltered
BLS	Shift B left one bit, zero to least significant bit. Sign unaltered
ARS	Shift A right one bit, extend sign;sign unaltered.
BRS	Shift B right one bit, extend sign;sign unaltered.
RAL	Rotate A left one bit
RBL	Rotate B left one bit
RAR	Rotate A right one bit
RBR	Rotate B right one bit
ALR	Shift A left one bit, clear sign, zero to least significant bit
BLR	Shift B left one bit, clear sign, zero to least significant bit
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	Skip next instruction if least significant bit in A is zero
SLB	Skip next instruction if least significant bit in B is zero

These instructions may be combined as follows:

label	$\left[\left\{\begin{matrix}\text{ALS}\\\text{ARS}\\\text{RAL}\\\text{RAR}\\\text{ALR}\\\text{ALF}\\\text{ERA}\\\text{ELA}\end{matrix}\right\}\right]$ $[\text{, CLE}]$ $[\text{, SLA}]$ $\left[,\left\{\begin{matrix}\text{ALS}\\\text{ARS}\\\text{RAL}\\\text{RAR}\\\text{ALR}\\\text{ALF}\\\text{ERA}\\\text{ELA}\end{matrix}\right\}\right]$	comments

label	$\left[\left\{\begin{matrix}\text{BLS}\\\text{BRS}\\\text{RBL}\\\text{RBR}\\\text{BLR}\\\text{BLF}\\\text{ERB}\\\text{ELB}\end{matrix}\right\}\right]$ $[\text{, CLE}]$ $[\text{, SLB}]$ $\left[,\left\{\begin{matrix}\text{BLS}\\\text{BRS}\\\text{RBL}\\\text{RBR}\\\text{BLR}\\\text{BLF}\\\text{ERB}\\\text{ELB}\end{matrix}\right\}\right]$	comments

CLE, SLA, or SLB appearing alone or in any valid combination with each other are assumed to be a Shift-Rotate machine instruction.

The Shift-Rotate instructions must be given in the order shown. At least one and up to four are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register.

No-Operation Instruction

When a no-operation is encountered in a program, no action takes place; the computer goes on to the next instruction. A full memory cycle is used in executing a no-operation instruction.

label	NOP	comments

A subroutine to be entered by a JSB instruction should have a

NOP as the first statement. The return address can be stored in the location occupied by the NOP during execution of the program. A NOP statement causes the Assembler to generate a word of zeros.

Alter-Skip Group

The Alter-Skip group contains 19 basic instructions that can be combined to produce more than 700 different single cycle operations.

CLA	Clear the A-Register
CLB	Clear the B-Register
CMA	Complement the A-Register
CMB	Complement the B-Register
CCA	Clear, then complement the A-Register (set to ones)
CCB	Clear, then complement the B-Register (set to ones)
CLE	Clear the E-Register
CME	Complement the E-Register
CCE	Clear, then complement the E-Register
SEZ	Skip next instruction if E is zero
SSA	Skip if sign of A is positive (0).
SSB	Skip if sign of B is positive (0).
INA	Increment A by one.
INB	Increment B by one.
SZA	Skip if contents of A equals zero
SZB	Skip if contents of B equals zero
SLA	Skip if least significant bit of A is zero
SLB	Skip if least significant bit of B is zero
RSS	Reverse the sense of the skip instructions. If no skip instructions precede in the statement, skip the next instruction.

These instructions may be combined as follows:

label	$\left[\begin{Bmatrix}\text{CLA}\\\text{CMA}\\\text{CCA}\end{Bmatrix}\right]$	[,SEZ]	,$\left[\begin{Bmatrix}\text{CLE}\\\text{CME}\\\text{CCE}\end{Bmatrix}\right]$	[,SSA] [,SLA] [,INA] [,SZA] [,RSS]	comments
label	$\left[\begin{Bmatrix}\text{CLB}\\\text{CMB}\\\text{CCB}\end{Bmatrix}\right]$	[,SEZ]	,$\left[\begin{Bmatrix}\text{CLE}\\\text{CME}\\\text{CCE}\end{Bmatrix}\right]$	[,SSB] [,SLB] [,INB] [,SZB] [,RSS]	comments

The Alter-Skip instructions must be given in the order shown. At least one and up to eight are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register. When two or more skip functions are combined in a single operation, a skip occurs if any one of the conditions exists. If a word with RSS also includes both SSA and SLA (or SSB and SLB) a skip occurs only when sign and least significant bit are both set (1).

3.3 INPUT/OUTPUT, OVERFLOW, AND HALT

The input/output instructions allow the user to transfer data to and from an external device via a buffer, to enable or disable external interrupt, or to check the status of I/O devices and operations. A subset of these instructions permits checking for an arithmetic overflow condition.

Input/Output instructions require the designation of a select code, sc, which indicates one of 64 input/output channels or functions. Each channel consists of a connect/disconnect control bit, a flag bit, and a buffer of up to 16 bits. The setting of the control bit indicates that a device associated with the channel is operable. The flag bit is set automatically when transmission between the device and the buffer is completed. Instructions are also available to test or clear the flag bit for the particular channel. If the interrupt system is enabled, setting of the flag causes program interrupt to occur; control transfers to the interrupt location related to the channel.

Expressions used to represent select codes (channel numbers) must have a value of less than 2^6. The value specifies the device or operation referenced. Instructions which transfer data between the A or B register and a buffer, access the Switch register when sc = 1. The character C appended to such an instruction clears the overflow bit after the transfer from the Switch register is complete.

Input/Output

Prior to any input/output data transmission, the control bit is set. The instruction which enables the device may also transfer data between the device and the buffer.

label	STC	sc [, C]	comments

Set I/O control bit for channel specified by sc. STC transfers or enables transfer of an element of data from an input device to the buffer or to an output device from the buffer. The exact function of the STC depends on the device; for the 2752A Teleprinter, an STC enables transfer of a series of bits. If sc = 1, this statement is treated as NOP. The C option clears the flag bit for the channel.

label	CLC	sc [, C]	comments

Clear I/O control bit for channel specified by sc. When the control bit is cleared, interrupt on the channel is disabled, although the flag may still be set by the device. If sc = 0, control bits for all channels are cleared to zero; all devices are disconnected. If sc = 1, this statement is treated as NOP.

label	LIA	sc [, C]	comments

Load into A the contents of the I/O buffer indicated by sc.

label	LIB	sc [, C]	comments

Load into B the contents of the I/O buffer indicated by sc.

label	MIA	sc [, C]	comments

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into A.

label	MIB	sc [,C]	comments

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into B.

label	OTA	sc [,C]	comments

Output the contents of A to the I/O buffer indicated by sc.

label	OTB	sc [,C]	comments

Output the contents of B to the I/O buffer indicated by sc.

label	STF	sc	comments

Sets the flag bit of the channel indicated by sc. If sc = 0, STF enables the interrupt system. A sc code of 1 causes the overflow bit to be set.

label	CLF	sc	comments

Clear the flag bit to zero for the channel indicated by sc. If sc = 0, CLF disables the interrupt system. If sc = 1, the overflow bit is cleared to zero.

label	SFC	sc	comments

Skip the next instruction if the flag bit for channel sc is clear. If sc = 1, the overflow bit is tested.

label	SFS	sc	comments

Skip the next instruction if the flag bit for channel sc is set. If sc = 1, the overflow is tested.

Overflow

In addition to the use of a select code of 1, the overflow bit may be accessed by the following instructions:

label	CLO	comments

Clear the overflow bit.

label	STO	comments

Set overflow bit.

label	SOC	[C]	comments

Skip the next instruction if the overflow bit is clear. The C option clears the bit after the test is performed.

label	SOS	[C]	comments

Skip the next instruction if the overflow bit is set. The C option clears the bit after the test is performed.

The C option is identified by the sequence 'space C space' following either 'SOC' or 'SOS'. Anything else is treated as a comment.

Halt

label	HLT	{[sc [,C]] [C]}	comments

Halt the computer. The machine instruction word is displayed in the T-Register. If the C option is used, the flag bit associated with channel sc is cleared.

If neither the select code nor the C option is used, the comments portion must be omitted.

3.4 EXTENDED ARITHMETIC INSTRUCTIONS

Ten instructions are used with the extended arithmetic version of the Assembler or Extended Assembler to increase the computer's overall efficiency. They provide for integer multiply and divide and for loading and storing double-length words to and from the accumulators.

label	MPY	m[,I] lit	comments

The MPY instruction multiplies the contents of the A-Register by the contents of m. The product is stored in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	m[,I] lit	comments

The DIV instruction divides the contents of registers B and A by the contents of m. The quotient is stored in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	DLD	m[,I] lit	comments

The DLD instruction loads the contents of locations m and m + 1 into registers A and B, respectively.

label	DST	m[,I]	comments

The DST instruction stores the contents of registers A and B in locations m and m + 1, respectively.

MPY, DIV, DLD, DST results in two machine words: a word for the instruction code and one for the operand.

The above four instructions are available without the extended arithmetic instructions as software subroutines.† However, by using the extended arithmetic group, they require less core storage and can be executed in less time.

The following shift-rotate instructions provide the capability to shift or rotate the B- and A-Registers n number of bit positions, where $1 \leqslant n \leqslant 16$.

label	ASR	n	comments

The ASR instruction arithmetically shifts the B- and A-Registers right n bits. The sign bit (bit 15 of B) is extended.

label	ASL	n	comments

The ASL instruction arithmetically shifts the B- and A-Register left n bits. Zeroes are placed in the least significant bits. The sign bit (bit 15 of B) is unaltered. The overflow bit is set if bit 14 differs from bit 15 before each shift, otherwise, exit with Overflow bit cleared.

label	RRR	n	comments

The RRR instruction rotates the B- and A-Registers right n bits.

label	RRL	n	comments

The RRL instruction rotates the B- and A-Registers left n bits.

label	LSR	n	comments

The LSR instruction logically shifts the B- and A-Registers right n bits. Zeroes are placed in the most significant bits.

label	LSL	n	comments

The LSL instruction logically shifts the B- and A-Registers left n bits. Place zeroes into the least significant bits.

† See ARITHMETIC SUBROUTINE CALLS, Section 4.7.

3.5 FLOATING-POINT INSTRUCTIONS

Floating-point instructions provide a means of performing calculations on floating-point values. Computers with the hardware floating-point option should use assemblers and libraries with floating-point capabilities. The floating-point assembler generates calls to the appropriate hardware function instead of the library subroutines. If the computer does not have the hardware floating-point option, then non-floating-point assemblers and libraries should be used.

```
FAD   { m [I,] }    comments
      { lit    }
```

FAD performs an addition between a floating-point number stored in the A- and B-Registers and a floating-point number stored in memory locations m and m + 1. The result is returned in the A- and B-Registers.

```
FSB   { m [I,] }    comments
      { lit    }
```

The FSB instruction subtracts a floating-point value in memory locations m and m + 1 from a floating-point value in the A- and B-Registers. The result is returned in the A- and B-Registers.

```
FMP   { m [I,] }    comments
      { lit    }
```

The FMP instruction multiplies a floating-point value in memory locations m and m + 1 with a floating-point value in the A- and B-Registers. The result is returned in the A- and B-Registers.

```
FDV   { m [I,] }    comments
      { lit    }
```

The FDV instruction divides the floating-point value in memory locations m and m + 1 into the value stored in the A- and B-Registers. The result is returned in the A- and B-Registers.

```
      FIX                   comments
```

The FIX instruction converts a floating-point number contained in the A-and B-Registers to a fixed point number. The result is returned in the A-Register. The contents of the B-Register are meaningless.

```
      FLT                   comments
```

The FLT instruction converts a fixed-point value contained in the A-Register to a floating-point value. The result is returned in the A- and B-Registers.

PSEUDO INSTRUCTIONS 4

The pseudo instructions control the Assembler, establish program relocatablility, and define program linkage as well as specify various types of constants, blocks of memory, and labels used in the program. With the Extended Assembler, pseudo instructions also control listing output.

4.1 ASSEMBLER CONTROL

The Assembler control pseudo instructions establish and alter the contents of the base page and program location counters, and terminate assembly processing. Labels may be used but they are ignored by the Assembler. NAM records produced by the Assemblers are accepted by the DOS, DOS-M and BCS Loaders.

	NAM	[name]	comments

NAM defines the name of a relocatable program. A relocatable program must begin with a NAM statement. † A relocatable program is assembled assuming a starting location of zero (i. e., zero relative). The name may be a symbol of one to five alphanumeric characters the first of which must be alphabetic or a period. The program name is printed on the list output. The name is optional and if omitted, the comments must be omitted also.

	ORG	m	comments

The ORG statement defines the origin of an absolute program, or the origin of subsequent sections of absolute or relocatable programs.

An absolute program must begin with an ORG statement. † The operand m, must be a decimal or octal integer specifying the initial setting of the program location counter.

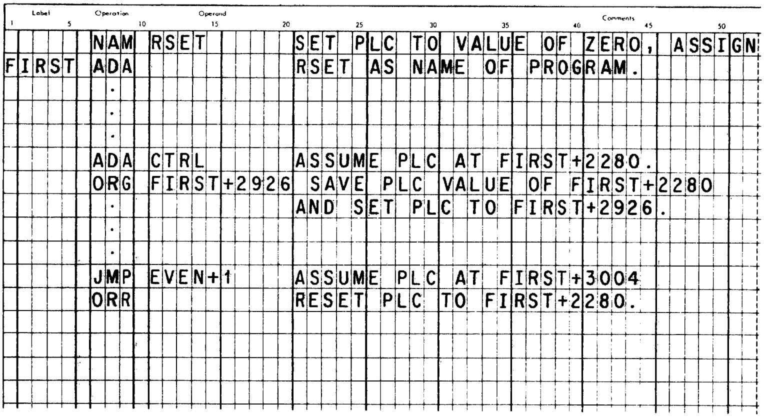
†The Control Statement, the HED instruction, and comments may appear prior to the NAM or ORG statements. If the Control Statement (ASMB,...) does not appear on tape preceding the program it must be entered from the Teleprinter.

ORG statements may be used elsewhere in the program to define starting addresses for portions of the object code. For absolute programs the Operand field, m, may be any expression. For relocatable programs, m, must be a program relocatable expression; it may not be base page or common relocatable or absolute. An expression is evaluated modulo 2^{15}. Symbols must be previously defined. All instructions following an ORG are assembled at consecutive addresses starting with the value of the operand.

	ORR	comments

ORR resets the program location counter to the value existing when an ORG or ORB instruction wås encountered.

Example:

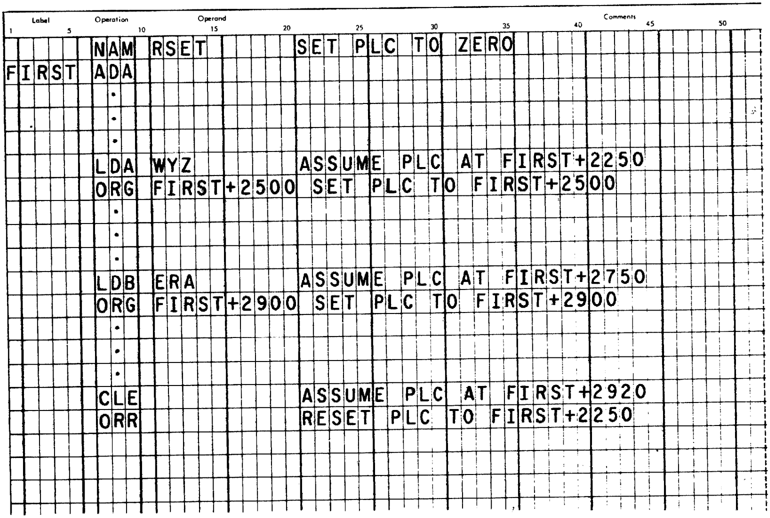


Label / Operation / Operand / Comments

```
      NAM RSET      SET PLC TO VALUE OF ZERO, ASSIGN
FIRST ADA           RSET AS NAME OF PROGRAM.
       .
       .
       .
      ADA CTRL      ASSUME PLC AT FIRST+2280.
      ORG FIRST+2926 SAVE PLC VALUE OF FIRST+2280
       .            AND SET PLC TO FIRST+2926.
       .
       .
      JMP EVEN+1    ASSUME PLC AT FIRST+3004
      ORR           RESET PLC TO FIRST+2280.
```

More than one ORG or ORB statement may occur before an ORR is used. If so, when the ORR is encountered, the program location counter is reset to the value it contained when the first ORG or ORB of the string occurred.

Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      NAM RSET      SET PLC TO ZERO
FIRST ADA
       .
       .
       .
      LDA WYZ       ASSUME PLC AT FIRST+2250
      ORG FIRST+2500 SET PLC TO FIRST+2500
       .
       .
       .
      LDB ERA       ASSUME PLC AT FIRST+2750
      ORG FIRST+2900 SET PLC TO FIRST+2900
       .
       .
       .
      CLE           ASSUME PLC AT FIRST+2920
      ORR           RESET PLC TO FIRST+2250
```

If a second ORR appears before an intervening ORG or ORB, the second ORR is ignored.

ORR cannot be used to reset the location counter for locations in the base page that are governed by the ORB statement.

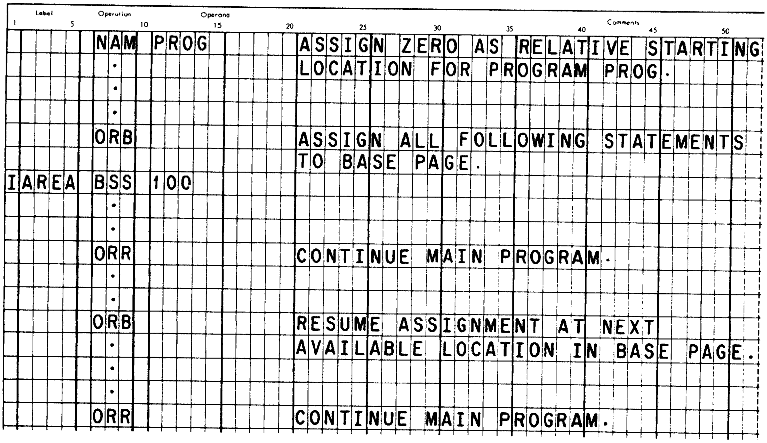
	ORB	comments

ORB defines the portion of a relocatable program that must be assigned to the base page by the Assembler. The Label field (if given) is ignored, and the statement requires no operand. All statements that follow the ORB statement are assigned contiguous locations in the base page. Assignment to the base page terminates when the Assembler detects an ORG, ORR, or END statement.

When more than one ORB is used in a program, each ORB causes the Assembler to resume assigning base page locations at the address following the last assigned base page location.

An ORB statement in an absolute program has no significance and is flagged as an error.

Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      NAM PROG      ASSIGN ZERO AS RELATIVE STARTING
       .            LOCATION FOR PROGRAM PROG.
       .
       .
      ORB           ASSIGN ALL FOLLOWING STATEMENTS
                    TO BASE PAGE.
IAREA BSS 100
       .
       .
      ORR           CONTINUE MAIN PROGRAM.
       .
       .
      ORB           RESUME ASSIGNMENT AT NEXT
       .            AVAILABLE LOCATION IN BASE PAGE.
       .
       .
      ORR           CONTINUE MAIN PROGRAM.
```

The IFN and IFZ pseudo instructions cause the inclusion of instructions in a program provided that either an "N" or "Z", respectively, is specified as a parameter for the ASMB control statement.† The IFN or IFZ instruction precedes the set of statements that are to be included. The pseudo instruction XIF serves as a terminator. If XIF is omitted, END acts as a terminator to both the set of statements and the assembly. IFN and IFZ may be used only when the source program is translated by the Extended Assembler which is provided for 8K or larger machines.

```
      IFN           comments
       .
       .
       .
      XIF
```

All source language statements appearing between the IFN and the XIF pseudo instructions are included in the program if the character "N" is specified on the ASMB control statement.

All source language statements appearing between the IFZ and the XIF pseudo instructions are included in the program if the character "Z" is specified on the ASMB control statement.

```
      IFZ           comments
       .
       .
       .
      XIF
```

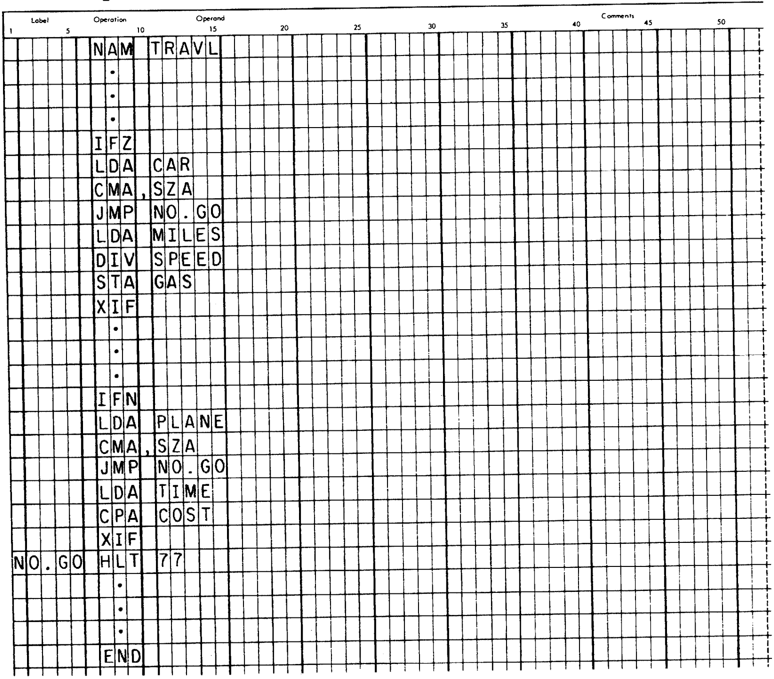
† See CONTROL STATEMENT, Section 5.1.

When the particular letter is not included on the control statement, the related set of statements appears on the Assembler output listing but is not assembled.

Any number of IFN-XIF and IFZ-XIF sets may appear in a program, however, they may not overlap. An IFZ or IFN intervening between an IFZ or IFN and the XIF terminator results in a diagnostic being issued during compilation; the second pseudo instruction is ignored.

Both IFN-XIF and IFZ-XIF pseudo instructions may be used in the program; however, only one type will be selected in a single assembly. Therefore, if both characters "N" and "Z" appear in the control statement, the character which is listed last will determine the set of coding that is to be included in the program.

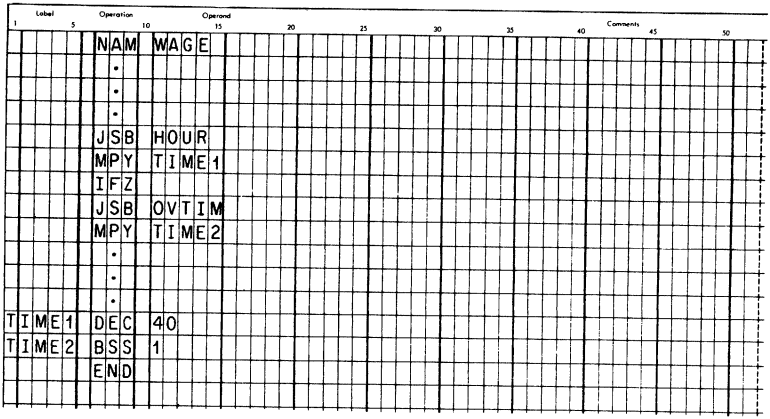
Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      NAM TRAVL
       .
       .
       .
      IFZ
      LDA CAR
      CMA,SZA
      JMP NO.GO
      LDA MILES
      DIV SPEED
      STA GAS
      XIF
       .
       .
       .
      IFN
      LDA PLANE
      CMA,SZA
      JMP NO.GO
      LDA TIME
      CPA COST
      XIF
NO.GO HLT 77
       .
       .
       .
      END
```

Program TRAVL will perform computations involving either or neither CAR or PLANE considerations depending on the presence or absence of Z or N parameters in the Control Statement.

Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      NAM  WAGE
       .
       .
       .
      JSB  HOUR
      MPY  TIME1
      IFZ
      JSB  OVTIM
      MPY  TIME2
       .
       .
       .
TIME1 DEC  40
TIME2 BSS  1
      END
```

Program WAGES computes a weekly wage value. Overtime consideration will be included in the program if "Z" is included in the parameters of the Control Statement.

The REP pseudo instruction, available in the Extended Assembler only, causes the repetition of the statement immediately following it a specified number of times.

label	REP	n	comments

The statement following the REP in the source program is repeated n times. The n may be any absolute expression. Comment lines (indicated by an asterisk in character position 1) are not repeated by REP. If a comment follows a REP instruction, the comment is ignored and the instruction following the comment is repeated.

A label specified in the REP pseudo instruction is assigned to the first repetition of the statement. A label cannot be part of the instruction to be repeated; it would result in a doubly defined symbol error.

Example:

```
         CLA
TRIPL    REP        3
         ADA        DATA
```

The above source code would generate the following:

```
                                Clear the A-Register;
         CLA                    the contents of DATA
TRIPL    ADA        DATA        is tripled and stored in
         ADA        DATA        the A-Register.
         ADA        DATA
```

Example:

```
FILL     REP        100B
         NOP
```

The example above loads 100_8 memory locations with the NOP instruction. The first location is labeled FILL.

Example:

```
         REP        2


         MPY        DATA
```

The above source code would generate the following:

```
         MPY        DATA
         MPY        DATA
```

```
    | END  | [m] |  comments
```

This statement terminates the program; it marks the physical end of the source language statements. The Operand field, m, may contain a name appearing as a statement label in the current program or it may be blank. If a name is entered, it identifies the location to which the loader transfers control after a relocatable program is loaded. A NOP should be stored at that location; the loader transfers control via a JSB.

If the Operand field is blank, the Comments field must be blank also, otherwise, the Assembler attempts to interpret the first five characters of the comments as the transfer address symbol.

The Label field of the END statement is ignored.

4.2 OBJECT PROGRAM LINKAGE

Linking pseudo instructions provide a means for communication between a main program and its subroutines or among several subprograms that are to be run as a single program. These instructions maybe used only in a relocatable program.

The Label field of this class is ignored in all cases. The Operand field is usually divided into many subfields, separated by commas. The first space not preceded by a comma or a left parenthesis terminates the entire field.

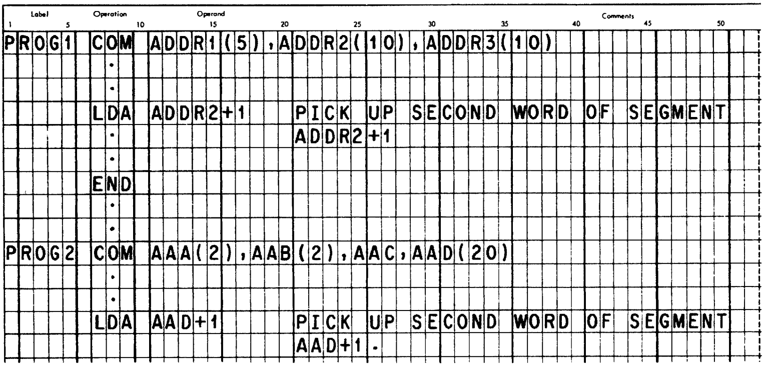
	COM	name, [(size$_1$)] [, name$_2$ [(size$_2$)], . . . , name$_n$[(size$_n$)]]	comments

COM reserves a block of storage locations that may be used in common by several subprograms. Each name identifies a segment of the block for the subprogram in which the COM statement appears. The sizes are the number of words allotted to the related segments. The size is specified as an octal or decimal integer. If the size is omitted, it is assumed to be one.

Any number of COM statements may appear in a subprogram. Storage locations are assigned contiguously; the length of the block is equal to the sum of the lengths of all segments named in all COM statements in the subprogram.

To refer to the common block, other subprograms must also include a COM statement. The segment names and sizes may be the same or they may differ. Regardless of the names and sizes specified in the separate subprograms, there is only one common block for the combined set. It has the same relative origin; the content of the nth word of common storage is the same for all subprograms.

Example:



```
PROG1 COM ADDR1(5),ADDR2(10),ADDR3(10)
       .
       .
      LDA ADDR2+1   PICK UP SECOND WORD OF SEGMENT
       .            ADDR2+1
       .
      END
       .
       .
PROG2 COM AAA(2),AAB(2),AAC,AAD(20)
       .
       .
      LDA AAD+1     PICK UP SECOND WORD OF SEGMENT
                    AAD+1.
```

Organization of common block:

PROG1 name	PROG2 name	Common Block
ADDR1	AAA	(location 1)
		(location 2)
	AAB	(location 3)
		(location 4)
	AAC	(location 5)
ADDR2	AAD	(location 6)
		(location 7)
		(location 8)
		(location 9)
		(location 10)
		(location 11)
		(location 12)
		(location 13)
		(location 14)
		(location 15)
ADDR3		(location 16)
		(location 17)
		(location 18)
		(location 19)
		(location 20)
		(location 21)
		(location 22)
		(location 23)
		(location 24)
		(location 25)

The LDA instructions in the two subprograms each refer to the same location in common storage, location 7.

The segment names that appear in the COM statements can be used in the Operand fields of DEF, ABS, EQU, or any Memory Reference statement; they may not be used as labels elsewhere in the program.

The loader establishes the origin of the common block; the origin cannot be set by the ORG or ORB pseudo instruction. All references to the common area are relocatable.

Two or more subprograms may declare common blocks which differ in size. The subprogram that defines the largest block must be the first submitted for loading.

	ENT	name$_1$ [, name$_2$, . . . , name$_n$]	comments

ENT defines entry points to the program or subprogram. Each name is a symbol that is assigned as a label for some machine operation in the program. Entry points allow another subprogram to refer to this subprogram. All entry points must be defined in the program.

Symbols appearing in an ENT statement may not also appear in EXT or COM statements in the same subprogram.

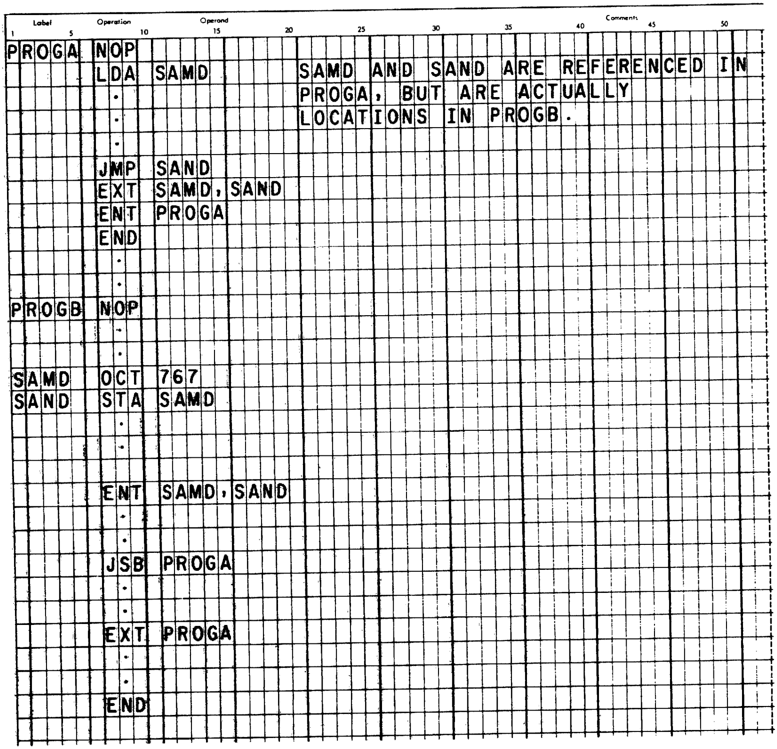
	EXT	name$_1$ [, name$_2$, . . . , name$_n$]	comments

This instruction designates labels in other subprograms which are referenced in this subprogram. The symbols must be defined as entry points by the other subprograms.

The symbols defined in the EXT statement may appear in Memory Reference statements, the EQU or DEF pseudo instructions. An external symbol must appear alone; it may not be in a multiple term expression or be specified as indirect. References to external locations are processed by the BCS loader as indirect addresses linked through the base page.

Symbols appearing in EXT statements may not also appear in ENT or COM statements in the same subprogram. The label field is ignored.

Example:



```
PROGA NOP
      LDA SAMD      SAMD AND SAND ARE REFERENCED IN
       .            PROGA, BUT ARE ACTUALLY
       .            LOCATIONS IN PROGB.
       .
      JMP SAND
      EXT SAMD,SAND
      ENT PROGA
      END
       .
       .
PROGB NOP
       .
       .
SAMD  OCT 767
SAND  STA SAMD
       .
       .
      ENT SAMD,SAND
       .
       .
      JSB PROGA
       .
       .
      EXT PROGA
       .
       .
      END
```

4.3 ADDRESS AND SYMBOL DEFINITION

The pseudo operations in this group assign a value or a word location to a symbol which is used as an operand elsewhere in the program.

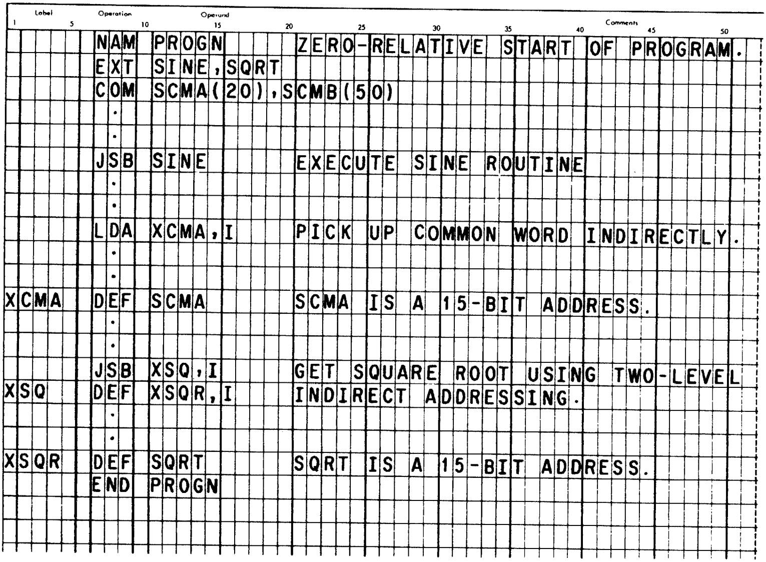
```
label | DEF | m [,I] | comments
```

The address definition statement generates one word of memory as a 15-bit address which may be used as the object of an indirect address found elsewhere in the source program. The symbol appearing in the label is that which is referenced; it appears in the Operand field of a Memory Reference instruction.

The operand field of the DEF statement may be any positive expression in an absolute program; in a relocatable program it may be a relocatable expression or an absolute expression with a value of less than 100_8. Symbols that do appear in the Operand field, may appear as operands of EXT or COM statements, in the same subprogram and as entry points in other subprograms.

The expression in the Operand field may itself be indirect and make reference to another DEF statement elsewhere in the source program.

Example:

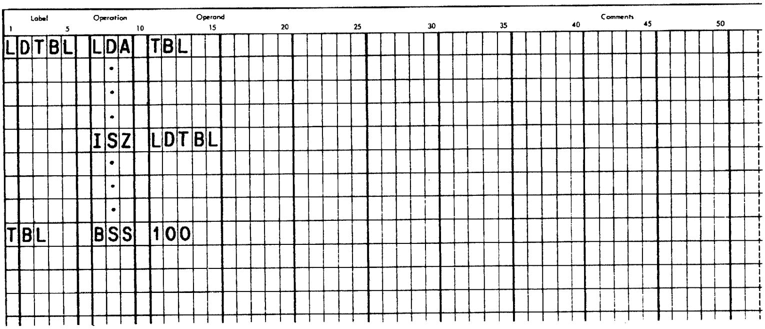

Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      NAM PROGN     ZERO-RELATIVE START OF PROGRAM.
      EXT SINE,SQRT
      COM SCMA(20),SCMB(50)
       .
       .
      JSB SINE      EXECUTE SINE ROUTINE
       .
       .
      LDA XCMA,I    PICK UP COMMON WORD INDIRECTLY.
       .
       .
XCMA  DEF SCMA      SCMA IS A 15-BIT ADDRESS.
       .
       .
      JSB XSQ,I     GET SQUARE ROOT USING TWO-LEVEL
XSQ   DEF XSQR,I    INDIRECT ADDRESSING.
       .
       .
XSQR  DEF SQRT      SQRT IS A 15-BIT ADDRESS.
      END PROGN
```

The DEF statement provides the necessary flexibility to perform address arithmetic in programs which are to be assembled in relocatable form. Relocatable programs should not modify the operand of a memory reference instruction.

In the example below, if TBL and LDTBL are in different pages, the BCS Loader processes TBL as an indirect address linked through the base page. The ISZ erroneously increments the loader provided reference to the base page rather than the value of TBL.

Example:



Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
LDTBL LDA TBL
       .
       .
       .
      ISZ LDTBL
       .
       .
       .
TBL   BSS 100
```

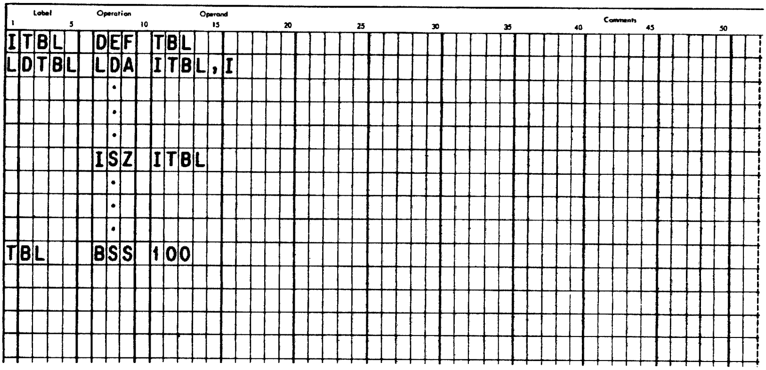
Assuming the loader might assign absolute locations comparable to the following octal values:

Page	Loc	Opcode	Reference
(0)	(700)	DEF	4000
		.	
		.	
		.	
(1)	(200)	LDA	(0) 700(I)
		.	
		.	
		.	
(1)	(300)	ISZ	(1) 200
		.	
		.	
		.	
(2)	(0)		(TBL)

It can be seen that the ISZ instruction would increment the quantity 700 rather than the address of the table (4000_8).

The following assures correct address modification during program execution.

Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
ITBL  DEF TBL
LDTBL LDA ITBL,I
       .
       .
       .
      ISZ ITBL
       .
       .
       .
TBL   BSS 100
```

This sequence might be stored by the loader as:

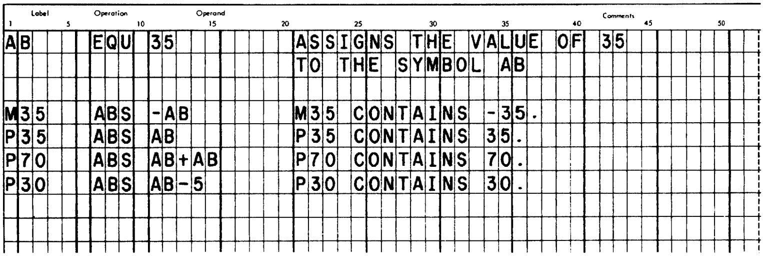
Page	Loc	Opcode	Reference
(1)	(200)	DEF	4000
(1)	(201)	LDA	200(I)
		.	
		.	
		.	
(1)	(300)	ISZ	(1) (200)
		.	
		.	
		.	
(2)	(0)		(TBL)

The value of 4000 is incremented; each execution of LDA will access successive locations in the table.

label	ABS	m	comments

ABS defines a 16-bit absolute value to be stored at the location represented by the label. The Operand field, m, may be any absolute expression; a single symbol must be defined as absolute elsewhere in the program.

Example:



```
AB    EQU 35        ASSIGNS THE VALUE OF 35
                    TO THE SYMBOL AB

M35   ABS -AB       M35 CONTAINS -35.
P35   ABS AB        P35 CONTAINS 35.
P70   ABS AB+AB     P70 CONTAINS 70.
P30   ABS AB-5      P30 CONTAINS 30.
```

label	EQU	m	comments

The EQU pseudo operation assigns to a symbol a value other than the one normally assigned by the program location counter. The symbol in the Label field is assigned the value represented by the Operand field. The Operand field may contain any expression. The value of the operand may be common, base page or program relocatable as well as absolute, but it may not be negative. Symbols appearing in the operand must be previously defined in the source program.

The EQU instruction may be used to symbolically equate two locations in memory; or it may be used to give a value to a symbol. The EQU statement does not result in a machine instruction.

Examples:

```
Label  Operation  Operand          Comments
1   5     10    15   20   25   30   35   40   45   50
      NAM FAM
       .
       .
       .
J3    DEF
       .
       .
       .
      LDA J3        THE SYMBOLS JFOUR AND J3+1 BOTH
      ADA ONE       IDENTIFY THE SAME LOCATION. THE
      STA J3+1      AND OPERATION IS PERFORMED ON
JFOUR EQU J3+1      THIS LOCATION.
       .
       .
       .
MWH   AND JFOUR
       .
       .
       .
```

Examples:

```
Label  Operation  Operand          Comments
1   5     10    15   20   25   30   35   40   45   50
      NAM STQTB
       .
       .
       .
      COM TABLA(1Ø) DEFINES A 1Ø WORD TABLE, TABLA.
       .
       .
       .
TABLB EQU TABLA+5   NAMES WORDS 6 THROUGH 1Ø OF
       .            TABLA AS TABLB.
       .
       .
       .
      LDA TABLB+1   LOADS CONTENTS OF 7TH WORD
       .            COMMON INTO A. THE STATEMENT LDA
       .            TABLA+6 WOULD PERFORM THE SAME
       .            OPERATION
       .
      NAM REG
       .
       .
       .
A     EQU Ø         DEFINES SYMBOL A AS Ø (LOCATION
B     EQU 1         OF A-REGISTER), AND SYMBOL B AS
       .            1(LOCATION OF B-REGISTER).
       .
      LDA B         LOADS CONTENTS OF B-REGISTER
                    INTO A-REGISTER.
```

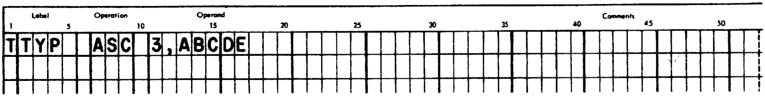
4.4 CONSTANT DEFINITION

The pseudo instructions in this class enter a string of one or more constant values into consecutive words of the object program. The statements may be named by labels so that other program statements can refer to the fields generated by them.

label	ASC	n, <2n characters>	comments

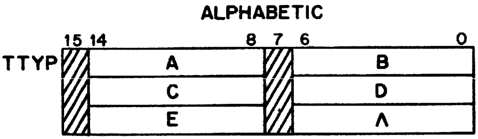
ASC generates a string of 2n alphanumeric characters in ASCII code into n consecutive words.† One character is right justified in each eight bits; the most significant bit is zero. n may be any expression resulting in an unsigned decimal value in the range 1 through 28. Symbols used in an expression must be previously defined. Anything in the Operand field following 2n characters is treated as comments. If less than 2n characters are detected before the end-of-statement mark, the remaining characters are assumed to be spaces, and are stored as such. The label represents the address of the first two characters.

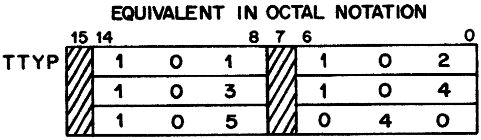
Example:



```
TTYP  ASC  3,ABCDE
```

causes the following:



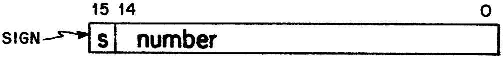


† To enter the code for the ASCII symbols which perform some action (e.g., (CR) and (LF)), the OCT pseudo instruction must be used.

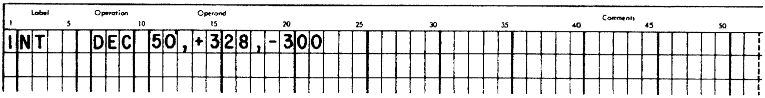
label	DEC	$d_1 [, d_2, \ldots, d_n]$	comments

DEC records a string of decimal constants into consecutive words. The constants may be either integer or real (floating point), and positive or negative. If no sign is specified, positive is assumed. The decimal number is converted to its binary equivalent by the Assembler. The label, if given, serves as the address of the first word occupied by the constant.

A decimal integer must be in the range of 0 to $2^{15}-1$; it may assume positive, negative, or zero values. It is converted into one binary word and appears as follows:



Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
INT  DEC 50,+328,-300
```

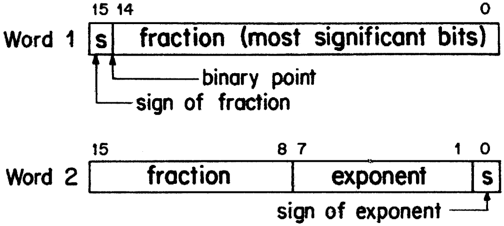
causes the following (octal representation)

	15	14				0
INT	0	0	0	0	6	2
	0	0	0	5	1	0
	1	7	7	3	2	4

A floating-point number has two components, a fraction and an exponent. The fraction is a signed or unsigned number which may be written with or without a decimal point. The exponent is indicated by the letter E and follows a signed or unsigned decimal integer. The floating-point number may have any of the following formats:

±n. n ±n. ±n. nE±e ±. nE±e ±n. E±e ±nE±e

The number is converted to binary, normalized (leading bits differ), and stored in two computer words. If either the fraction or the exponent is negative, that part is stored in two's complement form.



The floating-point number is made up of a 7-bit exponent with sign and a 23-bit fraction with sign. The number must be in the approximate range of 10^{-38} and zero.

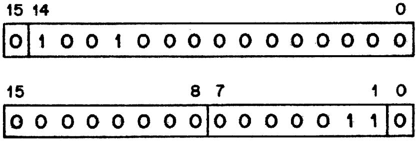
Examples:

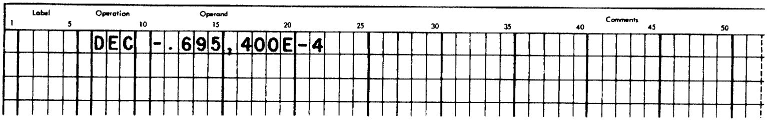
```
      DEC .45E1
      DEC 45.00E-1
      DEC 4500E-3
      DEC 4.5
```

are all equivalent to

.45×10^1

and are stored in normalized form as:





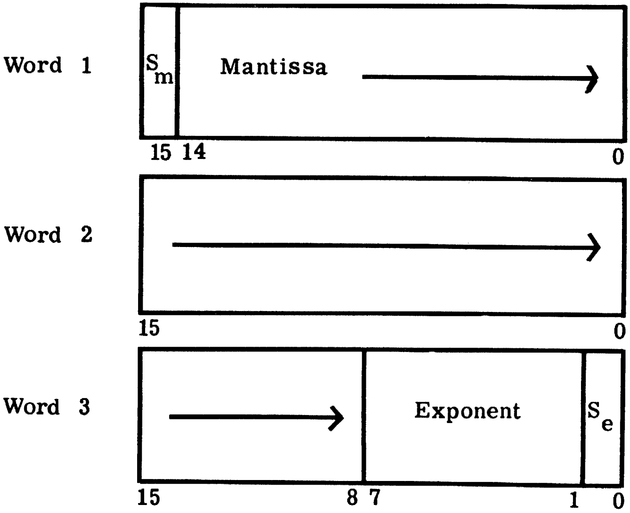
```
      DEC -.695,400E-4
```

are stored as:

```
|1|0 1 0 0 1 1 1 0 0 0 0 1 0 1 0|
|0 0 1 1 1 0 1 1|0 0 0 0 0 0 0|0|


|0|1 0 1 0 0 0 1 1 1 1 0 1 0 1 1|
|1 0 0 0 0 1 0 1|1 1 1 1 1 0 0|1|
```

label	DEX	$d_1[,d_2$,... , $d_n]$	comments

DEX, for the Extended Assembler, records a string of extended precision decimal constants into consecutive words within a program. Each such extended precision constant occupies three words as shown below:



Legend: S_m = Sign of the mantissa (fraction)

S_e = Sign of the Exponent

NOTE: a value is entered only if normalizing of the Mantissa is needed.

An extended precision floating-point number is made up of a 39-bit Mantissa (fraction) and sign and a 7-bit exponent and sign. The exponent and sign will be zero if the Mantissa does not have to be normalized.

This is the only form used for DEX. All values, whether they be floating-point, integer, fraction, or integer and fraction, will be stored in three words as just described. This storage format is basically an extension of that used for DEC, as previously described.

Examples:

DEX 12,-.45

are stored as:

WORD 1	WORD 2	WORD 3
0110000000000000	0000000000000000	0000000000001000

WORD 1	WORD 2	WORD 3
1000110011001100	1100110011001100	1001101111111111

label	OCT	o₁ [, o₂, . . . , oₙ]	comments

OCT stores one or more octal constants in consecutive words of the object program. Each constant consists of one to six octal digits (0 to 177777). If no sign is given, the sign is assumed to be positive. If the sign is negative, the two's complement of the binary equivalent is stored. The constants are separated by commas; the last constant is terminated by a space. If less than six digits are indicated for a constant, the data is right justified in the word. A label, if used, acts as the address of the first constant in the string. The letter B must not be used after the constant in the Operand field; it is significant only when defining an octal term in an instruction other than OCT.

Examples:

```
 Label     Operation   Operand                                      Comments
1   5         10          15        20        25        30        35        40        45        50
      OCT +Ø
      OCT -2
NUM   OCT 177,2Ø4Ø5,-36
      OCT 51,77777,-1,1Ø1Ø1
      OCT 1Ø7642,177Ø77
      OCT 1976                ILLEGAL: CONTAINS
      OCT -177777             DIGIT 9
      OCT 177B                ILLEGAL: CONTAINS
                              CHARACTER B
```

The previous statements are stored as follows:

	15	14				0
	0	0	0	0	0	0
	1	7	7	7	7	6
NUM	0	0	0	1	7	7
	0	2	0	4	0	5
	1	7	7	7	4	2
	0	0	0	0	5	1
	0	7	7	7	7	7
	1	7	7	7	7	7
	0	1	0	1	0	1
	1	0	7	6	4	2
	1	7	7	0	7	7
	X	X	X	X	X	X
	0	0	0	0	0	1
	X	X	X	X	X	X

THE RESULT OF ATTEMPTING TO DEFINE AN ILLEGAL CONSTANT IS UNPREDICTABLE

4.5 STORAGE ALLOCATION

The storage allocation statement reserves a block of memory for data or for a work area.

label	BSS	m	comments

The BSS pseudo operation advances the program or base page location counter according to the value of the operand. The Operand field may contain any expression that results in a positive integer. Symbols, if used, must be previously defined in the program. The label, if given, is the name assigned to the storage area and represents the address of the first word. The initial content of the area set aside by the statement is unaltered by the loader.

4.6 ASSEMBLY LISTING CONTROL

Assembly listing control pseudo instructions allow the user to control the assembly listing output during pass 2 or 3 of the assembly process. These pseudo instructions may be used only when the source program is translated by the Extended Assembler provided for 8K or larger machines (8, 192-word memory or larger).

	UNL	comments

Output is suppressed from the assembly listing, beginning with the UNL pseudo instruction and continuing for all instructions and comments until either an LST or END pseudo instruction is encountered. Diagnostic messages for errors encountered by the Assembler will be printed, however. The source statement sequence numbers (printed in columns 1-4 of the source program listing) are incremented for the instructions skipped.

	LST	comments

The LST pseudo instruction causes the source program listing, terminated by a UNL, to be resumed.

A UNL following a UNL, a LST following a LST, and a LST not preceded by a UNL are not considered errors by the Assembler.

	SUP	comments

The SUP pseudo instruction suppresses the output of additional code lines from the source program listing. Certain pseudo instructions, because they result in using subroutines, generate more than one line of coding. These additional code lines are suppressed by a SUP instruction until a UNS or the END pseudo instruction is encountered. SUP will suppress additional code lines in the following pseudo instructions:

ASC
OCT
DEC
DIV
DLD
DST
FAD
FDV
FMP
FSB
MPY

The SUP pseudo instruction may also be used to suppress the listing of literals at the end of the source program listing.

	UNS	comments

The UNS pseudo instruction causes the printing of additional coding lines, terminated by a SUP, to be resumed.

A SUP preceded by another SUP, UNS preceded by UNS, or UNS not preceded by a SUP are not considered errors by the Assembler.

	SKP	comments

The SKP pseudo instruction causes the source program listing to be skipped to the top of the next page. The SKP instruction is not listed, but the source statement sequence number is incremented for the SKP.

	SPC	n

The SPC pseudo instruction causes the source program listing to be skipped a specified number of lines. The list output is skipped n lines, or to the bottom of the page, whichever occurs first. The n may be any absolute expression. The SPC instruction is not listed but the source statement sequence number is incremented for the SPC.

	HED	m(heading)

The HED pseudo instruction allows the programmer to specify a heading to be printed at the top of each page of the source program listing.

The heading, m, a string of up to 56 ASCII characters, is printed at the top of each page of the source program listing following the occurrence of the HED pseudo instruction. If HED is encountered before the NAM or ORG at the beginning of a program, the heading will be used on the first page of the source program listing. A HED instruction placed elsewhere in the program causes a skip to the top of the next page.

The heading specified in the HED pseudo instruction will be used on every page until it is changed by a suceeding HED instruction.

The source statement containing the HED will not be listed, but source statement sequence number will be incremented.

4.7 ARITHMETIC SUBROUTINE CALLS

The members of this group of pseudo instructions request the Assember to generate calls to arithmetic subroutines† external to the source program. These pseudo instructions may be used in relocatable programs only. The Operand field may contain any relocatable expression or an absolute expression resulting in a value of less than 100_8.

label	MPY	{ m [,I] =Dn or =Bn }	comments

Multiply the contents of the A-register by the contents of m or the quantity defined by the literal and store the product in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	{ m [,I] =Dn or =Bn }	comments

Divide the contents of registers B and A by the contents of m or the quantity defined by the literal. Store the quotient in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	FMP	{ m [,I] =Fn }	comments

Multiply the two-word floating-point quantity in registers A and B by the two-word floating-point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating-point product in registers A and B.

label	FDV	{ m [,I] =Fn }	comments

Divide the two-word floating-point quantity in registers A and B by the two-word floating-point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating-point quotient in A and B.

† Not intended for use with DEX formatted numbers. For such numbers JSB's to double precision subroutines must be used. See Relocatable Subroutines Manual (02116-91780).

label	FAD	m [,I] =Fn	comments

Add the two-word floating point quantity in registers A and B to the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point sum in A and B.

label	FSB	m [,I] =Fn	comments

Subtract the two-word floating point quantity in m and m+1 or the quantity defined by the literal from the two-word floating point quantity in registers A and B and store the difference in A and B.

label	DLD	m [, I] =Fn	comments

Load the contents of locations m and m+1 or the quantity defined by the literal into registers A and B respectively.

label	DST	m [, I]	comments

Store the contents of registers A and B in locations m and m+1 respectively.

Each use of a statement from this group generates two words of instructions. Symbolically, they could be represented as follows:

```
JSB      <. arithmetic pseudo operation>
DEF      m [ , I]
```

An EXT <. arithmetic pseudo operation> is implied preceding the JSB operation.

In the above operations, the Overflow bit is set when one of the following conditions occurs:

Integer overflow
Floating-point overflow or underflow
Division by zero.

Execution of any of the subroutines alters the contents of the E-Register.

ASSEMBLER INPUT AND OUTPUT 5

The Assembler accepts as input a paper tape containing a control statement and a source language program. A relocatable source language program may be divided into several subroutines; the designation of these elements is optional. The output produced by the Assembler may include a punched paper tape containing the object program, an object program listing, and diagnostic messages.

5.1 CONTROL STATEMENT

The control statement specifies the output to be produced:

ASMB, p_1, p_2, . . . , p_n

''ASMB,'' is entered in positions 1-5. Following the comma are one or more parameters, in any order, which define the output to be produced. The control statement must be terminated by an end-of-statement mark, (CR) (LF).

The parameters may be any legal combination of the following starting in position 6:

A Absolute: The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity. It may not include NAM, ORB, COM, ENT, EXT, arithmetic pseudo operation statements or literals. The binary output format is that specified for the Basic Binary loader.

R Relocatable: The program may be located anywhere in memory. Instruction operands are adjusted as necessary. The binary output format is that specified for the BCS Relocating loader.

B Binary output: A program is to be punched according to one of the above parameters.

L List output: A program listing is to be produced either during pass two or pass three (if binary output selected) according to one of the above parameters.

T Table print: List the symbol table at the end of the first pass. For the Extended Assembler: List the symbol table in alphabetic order in three sections: section 1 for one- character symbols, section 2 for two- and three- character symbols, and section 3 for four- and five- character symbols.

N Include sets of instructions following the IFN pseudo instruction.

Z Include sets of instructions following the IFZ pseudo instruction.

F Acceptedby the Assembler to provide compatibility with DOSor DOS-M Assemblerprograms. F causes no action in any other assemblers.

(F = Extended Arithmetic Unit/Floating Point;
X = Nonextended Arithmetic Unit;
No parameter = Extended Arithmetic Unit)

Either A or R must be specified in addition to any combination of B, L, or T.

If a programmer wishes to assemble pass 1 of a source program to check for errors, he can specify only an A or R to be the sole parameter of the Assembler control statement, executing only pass 1. (This produces pass 1 error messages *without* listing the program or providing an object tape). Extended Assembler only.

The Assembler control statement must specifically request pass 2 operations (list or punch) in order for pass 2 to be executed. Lack of pass 2 option information causes processing only of pass 1 errors. If a C option is also provided, an automatic cross-reference symbol table is done after pass 1 when operating in the MTS environment.

The control statement may be on the same tape as the source program, or on a separate tape; or it may be entered via the Teleprinter keyboard.

5.2 SOURCE PROGRAM

The first statement of the program (other than remarks or a HED statement) must be a NAM statement for a relocatable program or an ORG statement for indicating the origin of an absolute program. The last statement must be an END statement and may contain a transfer address for the start of a relocatable program. Each statement is followed by an end-of-statement mark.

5.3 BINARY OUTPUT

The punch output is defined by the ASMB control statement. The punch output includes the instructions translated from the source program. It does not include system subroutines referenced within the source program (arithmetic subroutine calls, .IOC., .DIO., .ENTR, etc.)

5.4 LIST OUTPUT

Fields of the object program are listed in the following print columns.

Columns	Content
1-4	Source statement sequence number generated by the Assembler
5-6	Blank
7-11	Location (octal)
12	Blank
13-18	Object code word in octal
19	Relocation or external symbol indicator
20	Blank
21-72	First 52 characters of source statement.

Lines consisting entirely of comments (i.e., * in column 1) are printed as follows:

Columns	Content
1-4	Source statement sequence number
5-72	Up to 68 characters of comments

A Symbol Table listing has the following format:

Columns	Content
1-5	Symbol
6	Blank
7	Relocation of external symbol indicator
8	Blank
9-14	Value of the symbol

The characters that designate an external symbol or type of relocation for the Operand field or the symbol are as follows:

Character	Relocation Base
Blank	Absolute
R	Program relocatable
B	Base page relocatable
C	Common relocatable
X	External symbol

At the end of each pass, the following is printed:

```
** NO ERRORS*
   or
** nnnn ERRORS*
```

The value nnnn, indicates the number of errors.

NOTE: For complete operating instructions for the HP Assembler or extended Assembler, consult Software Operating Procedures, SIO Subsystems module (5931-1390).

HP CHARACTER SET A

ASCII CHARACTER FORMAT

b$_7$				0	0	0	0	1	1	1	1
b$_6$				0	0	1	1	0	0	1	1
b$_5$				0	1	0	1	0	1	0	1
b$_4$	b$_3$	b$_2$	b$_1$								
0	0	0	0	NULL	DC$_0$	ƀ	0	@	P	UNASSIGNED	UNASSIGNED
0	0	0	1	SOM	DC$_1$	!	1	A	Q		
0	0	1	0	EOA	DC$_2$	"	2	B	R		
0	0	1	1	EOM	DC$_3$	#	3	C	S		
0	1	0	0	EOT	DC4 (STOP)	$	4	D	T		
0	1	0	1	WRU	ERR	%	5	E	U		
0	1	1	0	RU	SYNC	&	6	F	V		
0	1	1	1	BELL	LEM	' (APOS)	7	G	W		
1	0	0	0	FE$_0$	S$_0$	(	8	H	X		
1	0	0	1	HT / SK	S$_1$	)	9	I	Y		
1	0	1	0	LF	S$_2$	*	:	J	Z		
1	0	1	1	V$_{TAB}$	S$_3$	+	;	K	[		
1	1	0	0	FF	S$_4$	, (COMMA)	<	L	\		ACK
1	1	0	1	CR	S$_5$	−	=	M	]		①
1	1	1	0	SO	S$_6$	.	>	N	↑		ESC
1	1	1	1	SI	S$_7$	/	?	O	←		DEL

Standard 7-bit set code positional order and notation are shown below with b$_7$ the high-order and b$_1$ the low-order, bit position.

	b$_7$	b$_6$	b$_5$	b$_4$	b$_3$	b$_2$	b$_1$
EXAMPLE: The code for "R" is:	1	0	1	0	0	1	0

LEGEND

NULL	Null/Idle
SOM	Start of message
EOA	End of address
EOM	End of message
EOT	End of transmission
WRU	"Who are you?"
RU	"Are you . . . ?"
BELL	Audible signal
FE$_0$	Format effector
HT	Horizontal tabulation
SK	Skip (punched card)
LF	Line feed
V$_{TAB}$	Vertical tabulation
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DC$_0$	Device control reserved for data link escape
DC$_1$-DC$_3$	Device Control
DC$_4$(Stop)	Device control (stop)
ERR	Error
SYNC	Synchronous idle
LEM	Logical end of media
S$_0$-S$_7$	Separator (information)
ƀ	Word separator (space, normally non-printing)
<	Less than
>	Greater than
↑	Up arrow (Exponentiation)
←	Left arrow (Implies/Replaced by)
\	Reverse slant
ACK	Acknowledge
①	Unassigned control
ESC	Escape
DEL	Delete/Idle

BINARY CODED DECIMAL FORMAT

Kennedy 1406/1506 ASCII-BCD Conversion

Symbol	BCD (octal code)	ASCII Equivalent (octal code)	Symbol	BCD (octal code)	ASCII Equivalent (octal code)
(Space)	2Ø	Ø4Ø	A	61	1Ø1
!	52	Ø41	B	62	1Ø2
#	13	Ø43	C	63	1Ø3
$	53	Ø44	D	64	1Ø4
%	34	Ø45	E	65	1Ø5
&	6Ø	Ø46	F	66	1Ø6
'	14	Ø47	G	67	1Ø7
(	34	Ø50	H	7Ø	11Ø
)	74	Ø51	I	71	111
*	54	Ø52	J	41	112
+	6Ø	Ø53	K	42	113
,	33	Ø54	L	43	114
-	4Ø	Ø55	M	44	115
.	73	Ø56	N	45	116
/	21	Ø57	O	46	117
			P	47	12Ø
Ø	12	Ø6Ø	Q	50	121
1	Ø1	Ø61	R	51	122
2	Ø2	Ø62	S	22	123
3	Ø3	Ø63	T	23	124
4	Ø4	Ø64	U	24	125
5	Ø5	Ø65	V	25	126
6	Ø6	Ø66	W	26	127
7	Ø7	Ø67	X	27	13Ø
8	1Ø	Ø7Ø	Y	30	131
9	11	Ø71	Z	31	132
:	15	Ø72	[	75	133
;	56	Ø73	\	36	134
<	76	Ø74	]	55	135
=	13	Ø75			
>	16	Ø76			
?	72	Ø77			
@	14	1ØØ			

Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

HP 2020A/B ASCII - BCD Conversion

Symbol	ASCII (Octal code)	BCD (Octal code)
(Space)	4Ø	2Ø
!	41	52
"	42	37
#	43	13
$	44	53
%	45	34
&	46	60 †
'	47	36
(	5Ø	75
)	51	55
*	52	54
+	53	6Ø
,	54	33
-	55	4Ø
.	56	73
/	57	21
Ø	6Ø	12
1	61	Ø1
2	62	Ø2
3	63	Ø3
4	64	Ø4
5	65	Ø5
6	66	Ø6
7	67	Ø7
8	7Ø	1Ø
9	71	11
:	72	15
;	73	56
<	74	76
=	75	35
>	76	16
?	77	72
@	1ØØ	14
A	1Ø1	61
B	1Ø2	62
C	1Ø3	63
D	1Ø4	64
E	1Ø5	65
F	1Ø6	66
G	1Ø7	67
H	11Ø	70
I	111	71
J	112	41
K	113	42
L	114	43
M	115	44
N	116	45
O	117	46
P	12Ø	47
Q	121	50
R	122	51
S	123	22
T	124	23
U	125	24
V	126	25
W	127	26
X	13Ø	27
Y	131	30
Z	132	31
[	133	75 ‡
]	135	55 ‡
↑	136	77
←	137	32

† BCD code of 60 always converted to ASCII code 53 (+).

‡ BCD code of 75 always converted to ASCII code 50 (() and BCD code of 55 always converted to ASCII code 51 ()).

ASSEMBLER INSTRUCTIONS B

Symbols	Meaning
label	Symbolic label, 1-5 alphanumeric characters and periods
m	Memory location represented by an expression
I	Indirect addressing indicator
C	Clear flag indicator
(m, m+1)	Two-word floating-point value in m and m+1
comments	Optional comments
[ ]	Optional portion of field
{ }	One of set may be selected
P	Program Counter
( )	Contents of location
∧	Logical product
∀	Exclusive "or"
∨	Inclusive "or"
A	A- register
B	B- register
E	E- register
A_n	Bit n of A-register
B_n	Bit n of B-register
b	Bit positions in B- and A-register
$\overline{(A/B)}$	Complement of contents of register A or B
(AB)	Two-word floating-point value in registers A and B
sc	Channel select code represented by an expression
d	Decimal constant
o	Octal constant
r	Repeat count
n	Integer constant
lit	Literal value

MACHINE INSTRUCTIONS

MEMORY REFERENCE

Jump and Increment-Skip

ISZ	m [,I]	(m) + 1 → m: then if (m) = 0, execute P + 2 otherwise execute P + 1
JMP	m [,I]	Jump to m; m → P
JSB	m [,I]	Jump subroutine to m: P + 1 → m; m + 1 → P

Add, Load and Store

ADA	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) + (A) → A
ADB	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) + (B) → B
LDA	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) → A
LDB	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) → B
STA	m [,I]	(A) → m
STB	m [,I]	(B) → m

Logical

AND	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) ∧ (A) → A
XOR	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) ⊻ (A) → A
IOR	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) ∨ (A) → A
CPA	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	If (m) ≠ (A), execute P + 2, otherwise execute P + 1
CPB	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	If (m) ≠ (B), execute P + 2, otherwise execute P + 1

REGISTER REFERENCE

Shift-Rotate

CLE	0 → E
ALS	Shift (A) left one bit, 0 → A_0, A_{15} unaltered
BLS	Shift (B) left one bit, 0 → B_0, B_{15} unaltered
ARS	Shift (A) right one bit, (A_{15}) → A_{14}
BRS	Shift (B) right one bit, (B_{15}) → B_{14}
RAL	Rotate (A) left one bit
RBL	Rotate (B) left one bit

Shift-Rotate (Continued)

RAR	Rotate (A) right one bit
RBR	Rotate (B) right one bit
ALR	Shift (A) left one bit, 0 → A_{15}
BLR	Shift (B) left one bit, 0 → B_{15}
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	If $(A_0) = 0$, execute P + 2, otherwise execute P + 1
SLB	If $(B_0) = 0$, execute P + 2, otherwise execute P + 1

Shift-Rotate instructions can be combined as follows:

$$\left[\begin{Bmatrix}\text{ALS}\\ \text{ARS}\\ \text{RAL}\\ \text{RAR}\\ \text{ALR}\\ \text{ALF}\\ \text{ERA}\\ \text{ELA}\end{Bmatrix}\right] \quad [,\text{CLE}] \quad [,\text{SLA}] \quad \left[,\begin{Bmatrix}\text{ALS}\\ \text{ARS}\\ \text{RAL}\\ \text{RAR}\\ \text{ALR}\\ \text{ALF}\\ \text{ERA}\\ \text{ELA}\end{Bmatrix}\right]$$

$$\left[\begin{Bmatrix}\text{BLS}\\ \text{BRS}\\ \text{RBL}\\ \text{RBR}\\ \text{BLR}\\ \text{BLF}\\ \text{ERB}\\ \text{ELB}\end{Bmatrix}\right] \quad [,\text{CLE}] \quad [,\text{SLB}] \quad \left[,\begin{Bmatrix}\text{BLS}\\ \text{BRS}\\ \text{RBL}\\ \text{RBR}\\ \text{BLR}\\ \text{BLF}\\ \text{ERB}\\ \text{ELB}\end{Bmatrix}\right]$$

No-operation

NOP	Execute P + 1

Alter-Skip

CLA	0's → A
CLB	0's → B
CMA	$\overline{(A)}$ → A
CMB	$\overline{(B)}$ → B
CCA	1's → A
CCB	1's → B
CLE	0 → E
CME	$\overline{(E)}$ → E

Alter-Skip (Continued)

CCE	1 → E
SEZ	If (E) = 0, execute P + 2, otherwise execute P + 1
SSA	If $(A_{15}) = 0$, execute P + 2, otherwise execute P + 1
SSB	If $(B_{15}) = 0$, execute P + 2, otherwise execute P + 1
INA	(A) + 1 → A
INB	(B) + 1 → B
SZA	If (A) = 0, execute P + 2, otherwise execute P + 1
SZB	If (B) = 0, execute P + 2, otherwise execute P + 1
SLA	If $(A_0) = 0$, execute P + 2, otherwise execute P + 1
SLB	If $(B_0) = 0$, execute P + 2, otherwise execute P + 1
RSS	Reverse sense of skip instructions. If no skip instructions precede, execute P + 2

Alter-Skip instructions can be combined as follows:

$$\left[\begin{Bmatrix}\text{CLA}\\ \text{CMA}\\ \text{CCA}\end{Bmatrix}\right]\ [,\text{SEZ}]\ \left[,\begin{Bmatrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{Bmatrix}\right]\ [,\text{SSA}]\ [,\text{SLA}]\ [,\text{INA}]\ [,\text{SZA}]\ [,\text{RRS}]$$

$$\left[\begin{Bmatrix}\text{CLB}\\ \text{CMB}\\ \text{CCB}\end{Bmatrix}\right]\ [,\text{SEZ}]\ \left[,\begin{Bmatrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{Bmatrix}\right]\ [,\text{SSB}]\ [,\text{SLB}]\ [,\text{INB}]\ [,\text{SZB}]\ [,\text{RSS}]$$

INPUT/OUTPUT, OVERFLOW, and HALT

Input/Output

STC	sc	[,C]	Set control bit_{sc}, enable transfer of one element of data be-tween device_{sc} and buffer_{sc}
CLC	sc	[,C]	Clear control bit_{sc}. If sc = 0 clear all control bits
LIA	sc	[,C]	(buffer_{sc}) → A
LIB	sc	[,C]	(buffer_{sc}) → B
MIA	sc	[,C]	$(\text{buffer}_{sc}) \vee (A)$ → A
MIB	sc	[,C]	$(\text{buffer}_{sc}) \vee (B)$ → B
OTA	sc	[,C]	(A) → buffer_{sc}
OTB	sc	[,C]	(B) → buffer_{sc}
STF	sc		Set flag bit_{sc}. If sc = 0, enable interrupt system. sc = 1 sets overflow bit.
CLF	sc		Clear flag bit_{sc}. If sc = 0, disable interrupt system. If sc =1, clear overflow bit.
SFC	sc		If $(\text{flag bit}_{sc}) = 0$, execute P + 2, otherwise execute P + 1. If sc = 1, test overflow bit.
SFS	sc		If $(\text{flag bit}_{sc}) = 1$, execute P + 2, otherwise execute P + 1. If sc = 1, test overflow bit.

Overflow

CLO		0 → overflow bit
STO		1 → overflow bit
SOC	[C]	If (overflow bit) = 0, execute P + 2, otherwise execute P + 1
SOS	[C]	If (overflow bit) = 0, execute P + 2, otherwise execute P + 1

Halt

HLT	[sc [,C]]	Halt computer

EXTENDED ARITHMETIC UNIT (requires EAU version of Assembler or Extender Assembler)

MPY	$\begin{Bmatrix} \text{m[,I]} \\ \text{lit} \end{Bmatrix}$	(A) x (m) → ($B_{\pm msb}$ and $A\|_{sb}$)
DIV	$\begin{Bmatrix} \text{m[,I]} \\ \text{lit} \end{Bmatrix}$	($B_{\pm msb}$ and $A\|_{sb}$)/(m) → A, remainder → B
DLD	$\begin{Bmatrix} \text{m[,I]} \\ \text{lit} \end{Bmatrix}$	(m) and (m + 1) → A and B
DST	$\begin{Bmatrix} \text{m[,I]} \\ \text{lit} \end{Bmatrix}$	(A) and (B) → m and m + 1
ASR	b	Arithmetically shift (BA) right b bits, B_{15} extended
ASL	b	Arithmetically shift (BA) left b bits, B_{15} unaltered, 0's to $A\|_{sb}$
RRR	b	Rotate (BA) right b bits
RRL	b	Rotate (BA) left b bits
LSR	b	Logically shift (BA) right b bits, 0's to B_{msb}
LSL	b	Logically shift (BA) left b bits, 0's to $A\|_{sb}$

PSEUDO INSTRUCTIONS

ASSEMBLER CONTROL

NAM	[name]	Specifies relocatable program and its name.
ORG	m	Gives absolute program origin or origin for a segment of relocatable or absolute program.
ORR		Reset main program location counter at value existing when first ORG or ORB of a string was encountered.
ORB		Defines base page portion of relocatable program.
END	[m]	Terminates source language program. Produces transfer to program starting location, m, if given.
REP < statement>	r	Repeat immediately following statement r times.
IFN < statements> XIF		Include statements in program if control statement contains N.
IFZ < statements> XIF		Include statements in program if control statement contains Z.

OBJECT PROGRAM LINKAGE

COM name$_1$ [(size$_1$)][, name$_2$ [(size$_2$)], . . . , name$_n$ [(size$_n$)]]

Reserves a block of common storage locations. name$_1$ identifies segments of block, each of length size.

ENT name$_1$ [, name$_2$, . . . , name$_n$]

Defines entry points, name$_1$, that may be referred to by other programs

EXT name$_1$ [, name$_2$, . . . , name$_n$]

Defines external locations, name$_1$, which are labels of other programs, referenced by this program.

ADDRESS AND SYMBOL DEFINITION

label	DEF	m[, I]	Generates a 15-bit address which may be referenced indirectly through the label.
label	ABS	m	Defines a 16-bit absolute value to be referenced by the label.
label	EQU	m	Equates the value, m, to the label.

CONSTANT DEFINITION

ASC	n, < 2n characters>	Generates a string of 2n ASCII characters.
DEC	d_1 [,d_2, . . . ,d_n]	Records a string of decimal constants of the form: Integer: ±n Floating-point: ±n.n, ±n., ±.n, ±nE±e, ±n.nE±e, ±n.E±e, ±.nE±e
DEX	d_1 [,d_2, ... , d_n]	Records a string of extended precision decimal constants of the form Floating point: ±n, ±n.n, ±n., ±.n, ±nE+e, ±n.nE±e, ±n.E+e, ±.nE±e
OCT	o_1 [,o_2, . . . ,o_n]	Records a string of octal constants of the form: ±oooooo

STORAGE ALLOCATION

BSS	m	Reserves a storage area of length, m.

ARITHMETIC SUBROUTINE CALLS REQUESTS††

MPY†	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	(A) x (m) → ($B_{\pm msb}$ and A_{lsb})
DIV†	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	($B_{\pm msb}$ and A_{lsb})/(m) → A, remainder → B
FMP	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	(AB) x (m, m + 1) → AB
FDV	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	(AB)/(m, m + 1) → AB
FAD	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	(m, m + 1) + (AB) → AB
FSB	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	(AB) - (m, m + 1) → AB
DLD†	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$	(m) and (m + 1) → A and B
DST†	m[,I]	(A) and (B) → m and m + 1

† For configurations including Extended Arithmetic Unit, these mnemonics generate hardware instructions when the EAU version of the Assembler or Extended Assembler is used.

††Not intended for use with DEX formatted numbers. For such numbers, JSB Machine Instructions must be used.

ASSEMBLY LISTING CONTROL

UNL		Suppress assembly listing output.
LST		Resume assembly listing output.
SKP		Skip listing to top of next page.
SPC	n	Skip n lines on listing
SUP		Suppress listing of extended code lines (e. g. , as produced by subroutine calls).
UNS		Resume listing of extended code lines.
HED	<heading>	Print <heading> at top of each page, where <heading> is up to 56 ASCII characters.

ALPHABETIC LIST OF INSTRUCTIONS C

ABS	Define absolute value
ADA	Add to A
ADB	Add to B
ALF	Rotate A left 4
ALR	Shift A left 1, clear sign
ALS	Shift A left 1
AND	"And" to A
ARS	Shift A right 1, sign carry
ASC	Generate ASCII characters
ASL	Arithmetic long shift left
ASR	Arithmetic long shift right
BLF	Rotate B left 4
BLR	Shift B left 1, clear sign
BLS	Shift B left 1
BRS	Shift B right 1, carry sign
BSS	Reserve block of storage starting at symbol
CCA	Clear and complement A (1's)
CCB	Clear and complement B (1's)
CCE	Clear and complement E (set E = 1)
CLA	Clear A
CLB	Clear B
CLC	Clear I/O control bit
CLE	Clear E
CLF	Clear I/O flag
CLO	Clear overflow bit
CMA	Complement A
CMB	Complement B

CME	Complement E
COM	Reserve block of common storage
CPA	Compare to A, skip if unequal
CPB	Compare to B, skip if unequal
DEC	Defines decimal constants
DEF	Defines address
DEX	Defines extended precision constants
DIV	Divide
DLD	Double load
DST	Double store
ELA	Rotate E and A left 1
ELB	Rotate E and B left 1
END	Terminate program
ENT	Entry point
ERA	Rotate E and A right 1
ERB	Rotate E and B right 1
EQU	Equate symbol
EXT	External reference
FAD	Floating add
FDV	Floating divide
FMP	Floating multiply
FSB	Floating subtract
HED	Print heading at top of each page
HLT	Halt
IFN	When N appears in Control Statement, assemble ensuing instructions
IFZ	When Z appears in Control Statement, assemble ensuing instructions
INA	Increment A by 1
INB	Increment B by 1
IOR	Inclusive "or" to A
ISZ	Increment, then skip if zero
JMP	Jump

JSB	Jump to subroutine
LDA	Load into A
LDB	Load into B
LIA	Load into A from I/O channel
LIB	Load into B from I/O channel
LSL	Logical long shift left
LSR	Logical long shift right
LST	Resume list output (follows a UNL)
MIA	Merge "or" into A from I/O channel
MIB	Merge "or" into B from I/O channel
MPY	Multiply
NAM	Names relocatable program
NOP	No operation
OCT	Defines octal constant
ORB	Establish origin in base page
ORG	Establish program origin
ORR	Reset program location counter
OTA	Output from A to I/O channel
OTB	Output from B to I/O channel
RAL	Rotate A left 1
RAR	Rotate A right 1
RBL	Rotate B left 1
RBR	Rotate B right 1
REP	Repeat next statement
RRL	Rotate A and B left
RRR	Rotate A and B right
RSS	Reverse skip sense
SEZ	Skip if E = 0
SFC	Skip if I/O flag = 0 (clear)
SFS	Skip if I/O flag = 1 (set)
SKP	Skip to top of next page

SLA	Skip if LSB of A = 0
SLB	Skip if LSB of B = 0
SOC	Skip if overflow bit = 0 (clear)
SOS	Skip if overflow bit = 1 (set)
SPC	Space n lines
SSA	Skip if sign A = 0
SSB	Skip if sign B = 0
STA	Store A
STB	Store B
STC	Set I/O control bit
STF	Set I/O flag
STO	Set overflow bit
SUP	Suppress list output of additional code lines
SWP	Switch the (A) and (B)
SZA	Skip if A = 0
SZB	Skip if B = 0
UNL	Suppress list output
UNS	Resume list output of additional code lines
XIF	Terminate an IFN or IFZ group of instructions
XOR	Exclusive "or" to A

SAMPLE PROGRAMS D

Following are two sample programs, the second of which implements several options of the Extended Assembler.

PARTS FILE UPDATE

A master file of parts is updated by a parts usage list to produce a new master parts file. A report, consisting of the parts used and their cost, is also produced.

The master file and the parts usage file contain four word records. Each record of the cost report is eleven words long.

The organization of the files is as follows:

Parts Master Files (PRTSM)

Identification	Quantity	Cost/ Item

Identification field of the Parts Master Files exists in ASCII althrough the entire record is read and written in binary.

Parts Usage File (PRTSU)

Identification	Quantity

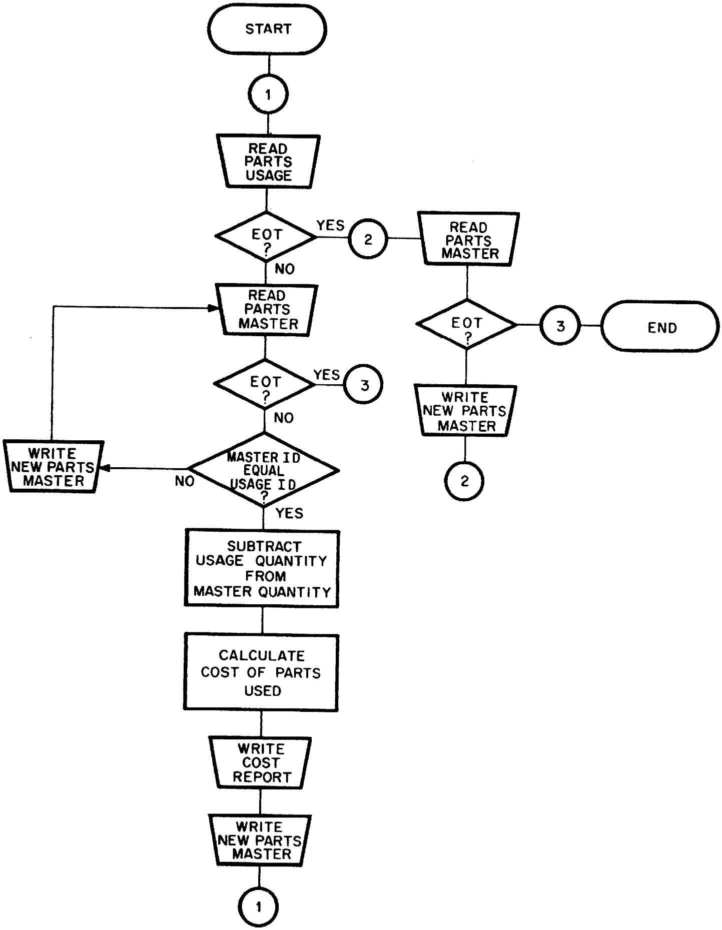
The parts usage file has been recorded in ASCII.

Parts Cost Report (PRTSC)

Identification	▨	Quantity used	▨	$	Cost for Quantity

The Parts Cost Report is recorded in ASCII with spacing and editing for printing.

The sample program reads and writes the files, adjusts the new stock levels, and calculates the cost. External subprograms perform the binary-to-decimal and decimal-to-binary conversions and handle unrecoverable input/output errors, invalid data conditions, and normal program termination. Input/output operations are performed using the Basic Control System input/output subroutine, .IOC.



SAMPLE PROGRAM
GENERAL FLOW CHART

SAMPLE ASSEMBLER SYMBOL TABLE OUTPUT

```
          0001     ASMB,R,B,L,T
START R 000000
PRTSM B 000000
PRTSU B 000004
PRTSC B 000010
EOTS1 B 000023
EOTS2 B 000024
MTEMP B 000025
UTEMP B 000026
SWTMP B 000027
SPACS B 000031
DLRSG B 000033
A       000000
B       000001
.IOC. X 000001
BCONV X 000002
DCONV X 000003
ABORT X 000004
HALT  X 000005
DTOBI C 000000
DTOBO C 000002
BTODI C 000003
BTODO C 000005
OPEN  R 000002
SPCFL R 000003
DLD   X 000006
DST   X 000007
READU R 000013
CKSTU R 000020
RJCTU R 000035
EOTU  R 000040
MSGU  R 000051
READM R 000063
CKSTM R 000070
RJCTM R 000105
EOTM  R 000110
MSGM  R 000117
HLTSW R 000137
COMPR R 000140
PROCM R 000157
PROCC R 000165
MPY   X 000010
CONVM R 000213
CONU1 R 000224
CONU2 R 000235
CONVC R 000246
WRITC R 000261
CKSTC R 000266
RJCTC R 000276
WRITN R 000301
CKSTN R 000306
RJCTN R 000316
**  NO ERRORS*
```

SAMPLE ASSEMBLER LIST OUTPUT

```
PAGE 0002

0001  00000               NAM UPDTE
0002  00000 000000  START NOP
0003  00001 026002R       JMP OPEN
0004  00000               ORB            ASSIGN STORAGE & CONSTANTS TO BP
0005  00000 000000  PRTSM BSS 4          MASTER PARTS FILE - BINARY.
0006  00004 000000  PRTSU BSS 4          PARTS USAGE LIST - ASCII.
0007  00010 000000  PRTSC BSS 11         PARTS COST REPORT - ASCII.
0008  00023 026063R EOTS1 JMP READM
0009  00024 026301R EOTS2 JMP WRITN
0010  00025 000000  MTEMP BSS 1
0011  00026 000000  UTEMP BSS 1
0012  00027 000000  SWTMP BSS 2
0013  00031 020040  SPACS ASC 2,
      00032 020040
0014  00033 020044  DLRSG ASC 1, $
0015  00000         A     EQU 0
0016  00001         B     EQU 1
0017                      EXT .IOC.      PERFORM I/O OPERATIONS USING BCS
0018*                                    I/O CONTROL ROUTINE.
0019                      EXT BCONV      ENTRY POINT FOR DECIMAL(ASCII)
0020*                                    TO BINARY CONVERSION SUBPROGRAM.
0021                      EXT DCONV      ENTRY POINT FOR BINARY TO
0022*                                    DECIMAL(ASCII) CONVERSION SUB-
0023*                                    PROGRAM.
0024                      EXT ABORT      ENTRY POINT FOR SUBPROGRAM WHICH
0025*                                    HANDLES UNRECOVERABLE I/O ERRORS
0026*                                    OR INVALID DATA.
0027                      EXT HALT       END OF PROGRAM SUBROUTINE.
0028                      COM DTOBI(2),DTOBO,BTODI(2),BTODO(2)
0029*                                    COMMON STORAGE LOCATIONS USED TO
0030*                                    PASS DATA BETWEEN MAIN PROGRAM
0031*                                    AND CONVERSION SUBPROGRAMS.
0032  00002               ORR            RESETS PLC AFTER USE OF ORB AT
0033*                                    BEGINNING OF PROGRAM.
0034  00002 000000  OPEN  NOP
0035  00003 016006X SPCFL DLD SPACS      STORES EDITING CHARACTERS IN
      00004 000031B
0036  00005 016007X       DST PRTSC+2    OUTPUT AREA FOR PARTS COST
      00006 000012B
0037  00007 016007X       DST PRTSC+6    REPORT.
      00010 000016B
0038  00011 060033B       LDA DLRSG
0039  00012 070020B       STA PRTSC+8
0040  00013 016001X READU JSB .IOC.      READ ONE RECORD FROM USAGE LIST
0041  00014 010001        OCT 10001      LOCATED ON STANDARD UNIT 1
0042  00015 026035R       JMP RJCTU      (TELEPRINTER INPUT). PRTSU IS
0043  00016 000004B       DEF PRTSU      ADDRESS OF STORAGE AREA; AREA IS
0044  00017 000004        DEC 4          4 WORDS LONG.
0045  00020 016001X CKSTU JSB .IOC.      CHECK STATUS OF UNIT 1.
0046  00021 040001        OCT 40001
0047  00022 002020        SSA
0048  00023 026020R       JMP CKSTU      IF BUSY, LOOP UNTIL FREE.
0049  00024 001200        RAL
0050  00025 002020        SSA
0051  00026 026030R       JMP *+2
0052  00027 026063R       JMP READM      IF COMPLETE, TRANSFER TO SECTION
0053*                                    WHICH READS MASTER FILE RECORD.
```

```
PAGE 0003


0054  00030 001727        ALF,ALF        TEST END OF TAPE STATUS BIT
0055  00031 001200        RAL            (ORIGINAL BIT 05).
0056  00032 002020        SSA
0057  00033 026040R       JMP EOTU       IF SET, GO TO EOT PROCEDURE.
0058  00034 026004X       JMP ABORT      IF NOT SET, SOME ERROR CONDITION
0059*                                    (UNRECOVERABLE) EXISTS.
0060  00035 006020  RJCTU SSB            CHECK CAUSE OF REJECT. IF UNIT
0061  00036 026013R       JMP READU      BUSY LOOP UNTIL FREE. ANY OTHER
0062  00037 026004X       JMP ABORT      CAUSE IS UNRECOVERABLE ERROR.
0063  00040 060023B EOTU  LDA EOTS1      IF END OF USAGE FILE, ALTER
0064  00041 072002R       STA OPEN       PROGRAM SEQUENCE TO BYPASS
0065  00042 060024B       LDA EOTS2      SECTIONS THAT READ AND PROCESS
0066  00043 072140R       STA COMPR      USAGE FILE. PRINT MESSAGE ON
0067  00044 016001X       JSB .IOC.      TELEPRINTER INDICATING EOT.
0068  00045 020002        OCT 20002
0069  00046 026044R       JMP EOTU+4
0070  00047 000051R       DEF MSGU
0071  00050 000011        DEC 9
0072  00051 042516  MSGU  ASC 9,END OF USAGE FILE
      00052 042040
      00053 047506
      00054 020125
      00055 051501
      00056 043505
      00057 020106
      00060 044514
      00061 042440
0073  00062 026063R       JMP READM
0074  00063 016001X READM JSB .IOC.      READ A RECORD FROM MASTER PARTS
0075  00064 010105        OCT 10105      FILE ON STANDARD UNIT 05(PUNCHED
0076  00065 026105R       JMP RJCTM      TAPE READER). PRTSM  IS ADDRESS
0077  00066 000000B       DEF PRTSM      OF STORAGE AREA; AREA IS 4 WORDS
0078  00067 000004        DEC 4          LONG. RECORD IS IN BINARY FORMAT
0079  00070 016001X CKSTM JSB .IOC.      CHECK STATUS OF UNIT 5.
0080  00071 040005        OCT 40005
0081  00072 002020        SSA
0082  00073 026070R       JMP CKSTM      IF BUSY, LOOP UNTIL FREE.
0083  00074 001200        RAL
0084  00075 002020        SSA
0085  00076 026100R       JMP *+2
0086  00077 026140R       JMP COMPR      IF COMPLETE, TRANSFER TO EITHER
0087  00100 001727        ALF,ALF        PROCESSING OR WRITE OUTPUT
0088  00101 001200        RAL            DEPENDING ON SETTING OF COMPR.
0089  00102 002020        SSA            TEST FOR END OF TAPE.
0090  00103 026110R       JMP EOTM       IF END, GO TO EOT PROCEDURE.
0091  00104 026004X       JMP ABORT      IF NOT, AN UNRECOVERABLE ERROR
0092*                                    EXISTS.
0093  00105 006020  RJCTM SSB            CHECK CONTENTS OF B FOR CAUSE OF
0094  00106 026063R       JMP READM      REJECT. IF UNIT BUSY, LOOP UNTIL
0095  00107 026004X       JMP ABORT      FREE, OTHERWISE I/O ERROR EXISTS
0096  00110 062137R EOTM  LDA HLTSW      ALTER PROGRAM SEQUENCE TO HALT
0097  00111 072315R       STA CKSTN+7    EXECUTION AFTER LAST RECORD IS
0098  00112 016001X       JSB .IOC.      WRITTEN  PRINT MESSAGE
0099  00113 020002        OCT 20002      INDICATING END OF MASTER INPUT.
0100  00114 026112R       JMP EOTM+2
0101  00115 000117R       DEF MSGM
0102  00116 000017        DEC 15
0103  00117 042516  MSGM  ASC 15,END OF MASTER PARTS FILE INPUT
```

```
PAGE 0004

      00120 042040
      00121 047506
      00122 020115
      00123 040523
      00124 052105
      00125 051040
      00126 050101
      00127 051124
      00130 051440
      00131 043111
      00132 046105
      00133 020111
      00134 047120
      00135 052524
0104  00136 026140R       JMP COMPR
0105  00137 026005X HLTSW JMP HALT       END OF PROGRAM SUBROUTINE.
0106  00140 000000  COMPR NOP
0107  00141 016224R       JSB CONU1      CONVERT ID NUMBER FIELDS OF
0108  00142 016213R       JSB CONVM      MASTER AND USAGE FILES TO BIN.
0109  00143 060026B       LDA UTEMP      LOAD THESE FIELDS FROM TEMPORARY
0110  00144 064025B       LDB MTEMP      STORAGE.
0111  00145 050001        CPA B          COMPARE
0112  00146 026157R       JMP PROCM      IF EQUAL, JUMP TO PROCESSING
0113  00147 007004        CMB,INB        IF ID NUMBER OF MASTER GREATER
0114  00150 040001        ADA B          THAN ID NUMBER OF USAGE, DATA IN
0115  00151 002020        SSA            USAGE FILE ERRONEOUS. TERMINATE
0116  00152 026004X       JMP ABORT      RUN.
0117  00153 062156R       LDA *+3        IF ID MASTER LESS THAN ID USAGE,
0118  00154 072315R       STA CKSTN+7    ALTER SEQUENCE: READ NEXT MASTER
0119  00155 026301R       JMP WRITN      RECORD IMMEDIATELY AFTER WRITING
0120  00156 026063R       JMP READM      CURRENT MASTER RECORD.
0121  00157 016235R PROCM JSB CONU2      CONVERT QUANTITY FIELD OF USAGE
0122  00160 060002B       LDA PRTSM+2    FILE TO BINARY AND SUBTRACT FROM
0123  00161 064027B       LDB UTEMP+1    QUANTITY FIELD OF MASTER AND
0124  00162 007004        CMB,INB        STORE RESULT.
0125  00163 040001        ADA B
0126  00164 070002B       STA PRTSM+2
0127  00165 016006X PROCC DLD PRTSU      STORE ID OF PARTS USED IN REPORT
      00166 000004B
0128  00167 016007X       DST PRTSC      FILE STORAGE AREA.
      00170 000010B
0129  00171 016006X       DLD PRTSU+2    STORE QUANTITY OF PARTS USED IN
      00172 000006B
0130  00173 016007X       DST PRTSC+4    REPORT FILE STORAGE AREA.
      00174 000014B
0131  00175 060003B       LDA PRTSM+3    COMPUTE COST OF PARTS USED.
0132  00176 016010X       MPY UTEMP+1
      00177 000027B
0133  00200 070030B       STA SWTMP+1
0134  00201 074027B       STB SWTMP
0135  00202 016246R       JSB CONVC      CONVERT RESULT TO DECIMAL
0136  00203 016006X       DLD SWTMP
      00204 000027B
0137  00205 016007X       DST PRTSC+9    STORE IN REPORT FILE AREA.
      00206 000021B
0138  00207 062212R       LDA *+3        ALTER SEQUENCE: READ NEXT USAGE
0139  00210 072315R       STA CKSTN+7    RECORD AFTER WRITING CURRENT
0140  00211 026261R       JMP WRITC      MASTER RECORD.
```

```
0141  00212 026013R        JMP READU
0142  00213 000000   CONVM NOP
0143  00214 016006X        DLD PRTSM        STORE ID FIELDS IN COMMON
      00215 000000B
0144  00216 016007X        DST DTOBI        LOCATIONS TO BE PROCESSED BY
      00217 000000C
0145  00220 016002X        JSB BCONV        CONVERSION SUBPROGRAM. ON
0146  00221 062002C        LDA DTOBO        COMPLETION, STORE RESULTS IN
0147  00222 070025B        STA MTEMP        LOCATIONS USED BY PROCESSING
0148  00223 126213R        JMP CONVM,I      SECTIONS. CONVM APPLIES TO ID OF
0149  00224 000000   CONU1 NOP              MASTER PARTS FILE; CONU1, TO ID
0150  00225 016006X        DLD PRTSU        OF USAGE; CONU2, TO QUANTITY OF
      00226 000004B
0151  00227 016007X        DST DTOBI        USAGE; AND CONVC, TO COST OF
      00230 000000C
0152  00231 016002X        JSB BCONV        PARTS(THIS IS A BINARY TO
0153  00232 062002C        LDA DTOBO        DECIMAL CONVERSION).
0154  00233 070026B        STA UTEMP
0155  00234 126224R        JMP CONU1,I
0156  00235 000000   CONU2 NOP
0157  00236 016006X        DLD PRTSU+2
      00237 000006B
0158  00240 016007X        DST DTOBI
      00241 000000C
0159  00242 016002X        JSB BCONV
0160  00243 062002C        LDA DTOBO
0161  00244 070027B        STA UTEMP+1
0162  00245 126235R        JMP CONU2,I
0163  00246 000000   CONVC NOP
0164  00247 016006X        DLD SWTMP
      00250 000027B
0165  00251 016007X        DST BTODI
      00252 000003C
0166  00253 016003X        JSB DCONV
0167  00254 016006X        DLD BTODO
      00255 000005C
0168  00256 016007X        DST SWTMP
      00257 000027B
0169  00260 126246R        JMP CONVC,I
0170  00261 016001X WRITC JSB .IOC.        WRITE ONE RECORD OF PARTS COST
0171  00262 020102         OCT 20102        REPORT ON STANDARD UNIT 2
0172  00263 026276R        JMP RJCTC        (TELEPRINTER OUTPUT). PRTSC IS
0173  00264 000010B        DEF PRTSC        ADDRESS IN STORAGE AREA; AREA IS
0174  00265 000013         DEC 11           11 WORDS LONG. RECORD IS IN ASCI
0175  00266 016001X CKSTC JSB .IOC.        CHECK STATUS OF UNIT 2.
0176  00267 040002         OCT 40002
0177  00270 002020         SSA
0178  00271 026266R        JMP CKSTC        IF BUSY, LOOP UNTIL FREE.
0179  00272 001200         RAL
0180  00273 002020         SSA
0181  00274 026004X        JMP ABORT        TERMINATE IF ANY I/O ERROR.
0182  00275 026301R        JMP WRITN        IF COMPLETE, TRANSFER TO WRITN.
0183  00276 006020  RJCTC  SSB              IF BUSY, LOOP UNTIL FREE.
0184  00277 026261R        JMP WRITC        TERMINATE ON ANY OTHER REJECT
0185  00300 026004X        JMP ABORT        CONDITION.
0186  00301 016001X WRITN JSB .IOC.        WRITE ONE RECORD (BINARY) OF
0187  00302 020104         OCT 20104        NEW MASTER PARTS LIST ON UNIT 4
0188  00303 026316R        JMP RJCTN        (TAPE PUNCH). PRTSM (INPUT AREA)
```

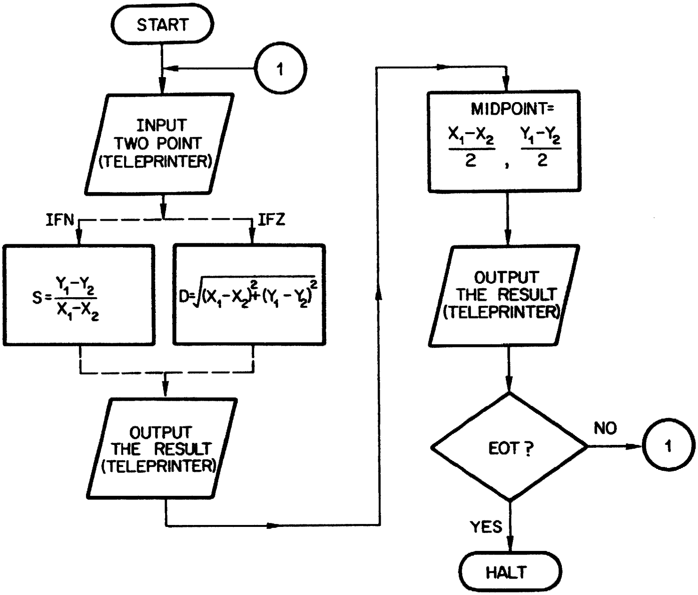
```
 PAGE 0006

0189  00304 000000B        DEF PRTSM     IS ALSO USED AS OUTPUT AREA.
0190  00305 000004         DEC 4
0191  00306 016001X CKSTN  JSB .IOC.     CHECK STATUS OF UNIT 4.
0192  00307 040004         OCT 40004
0193  00310 002020         SSA
0194  00311 026306R        JMP CKSTN     IF BUSY, LOOP UNTIL FREE.
0195  00312 001200         RAL
0196  00313 002020         SSA
0197  00314 026004X        JMP ABORT
0198  00315 026013R        JMP READU
0199  00316 006020  RJCTN  SSB           IF BUSY, LOOP UNTIL FREE, OTHER-
0200  00317 026301R        JMP WRITN     WISE TERMINATE.
0201  00320 026004X        JMP ABORT
0202                       END START
**   NO ERRORS*
```

CALCULATING DISTANCE

Program "Line" will either calculate the distance between two points or find the slope of the line connecting the points; then the point equidistant from each point (the mid-point) is calculated.

Data is input using the formatter library routine four n-digit real numbers at a time. The first quantity is the X coordinate of the first point; the second quantity is the Y coordinate of the first point; the third and fourth quantities are the X and Y coordinates of the second point.

The result is output to the teleprinter by the formatter library routine; each quantity cannot be more than an eight digit real number.


START
1
INPUT TWO POINT (TELEPRINTER)
IFN
IFZ
$S=\frac{Y_1-Y_2}{X_1-X_2}$
$D=\sqrt{(X_1-X_2)^2+(Y_1-Y_2)^2}$
OUTPUT THE RESULT (TELEPRINTER)
MIDPOINT= $\frac{X_1-X_2}{2}$, $\frac{Y_1-Y_2}{2}$
OUTPUT THE RESULT (TELEPRINTER)
EOT ?
NO
1
YES
HALT

GENERAL FLOW CHART

Below is the source program as it is typed up on the teleprinter. After it are the assembler listings. The first listing results from including the Z option in the control statement. In the second listing the N option has been included in the control statement.

> NOTE: When the complete data tape has been read and the tape reader encounters 10 blank feed frames, an EQT message is typed on the teleprinter and the computer halts. Thus no halt instruction is needed in the program.)

```
      HED LINE FORMULI:  DISTANCE, SLOPE, MID-POINT
*  PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
*  TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
*  THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
*  POINT (THE MID-POINT) IS CALCULATED.
*     DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
*  FOUR N-DIGIT REAL NUMBERS AT A TIME.  THE FIRST
*  QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
*  SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
*  THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
*  OF THE SECOND POINT.
*     THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
*  FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
*  THAN AN EIGHT DIGIT REAL NUMBER.
      NAM LINE
START NOP
      JMP INPUT
      EXT .IOC.,FLOAT,IFIX,SQRT
      EXT .DIO.,.IOI.,.DTA.,.RAR.
      EXT .IOR.,.IAR.
.DATA DEF DATA
.PRIN DEF PRINT
DATA  BSS 4
FMT   ASC 3,(F8.3)
FMT2  ASC 8,(F8.3,",",F8.3/)
FMT3  ASC 3,(4I2)
      SKP
*  INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
INPUT NOP
      LDA =B5
      CLB,INB
      JSB .DIO.
      DEF FMT3
      DEF *+4
      LDA =B4
      LDB .DATA
      JSB .IAR.
      SPC 3
*  THE DISTANCE BETWEEN THE TWO POINTS:
      IFZ
      LDA DATA+2
      CMA,INA
      ADA DATA
      SPC 1
      JMP *+5
PRINT REP 4
      NOP
      SPC 1
      STA PRINT
      SUP
      MPY PRINT
      STA PRINT
      SPC 1
      LDA DATA+3
      CMA,INA
      ADA DATA+1
      STA PRINT+1
      MPY PRINT+1
      ADA PRINT
      SPC 1
      JSB FLOAT
      JSB SQRT
      DST PRINT
      XIF
      SPC 3
*  FIND THE SLOPE OF THE LINE
      IFN
      LDA DATA+2
      CMA,INA
      ADA DATA
      JMP *+5
PRINT REP 4
      NOP
      STA PRINT
      SPC 1
      LDA DATA+3
      CMA,INA
      ADA DATA+1
      CLB
      DIV PRINT
      DST PRINT
      XIF
      SPC 3
*  OUTPUT THE RESULT
      LDA =B2
      CLB
      JSB .DIO.
      DEF FMT
      DEF *+4
      DLD PRINT
      JSB .IOR.
      JSB .DTA.
      SPC 3
*  FIND THE MID-POINT OF THE LINE SEGMENT:
      LDA DATA
      ADA DATA+2
      CLB
      JSB FLOAT
      FMP =F.5
      DST PRINT
      SPC 1
      LDA DATA+1
      ADA DATA+3
      CLB
      JSB FLOAT
      FMP =F.5
      DST PRINT+2
      SPC 1
      UNL
```

```
LDA =B2
CLB
JSB .DIO.
DEF FMT2
DEF *+5
LDA =B2
LDB .PRIN
JSB .RAR.
JSB .DTA.
LST
SPC 3
UNS
JMP INPUT
END START
```

```
 PAGE 0001

0001                     ASMB,R,L,T,Z
START R 000000
.IOC. X 000001
FLOAT X 000002
IFIX  X 000003
SQRT  X 000004
.DIO. X 000005
.IOI. X 000006
.DTA. X 000007
.RAR. X 000010
.IOR. X 000011
.IAR. X 000012
.DATA R 000002
.PRIN R 000003
DATA  R 000004
FMT   R 000010
FMT2  R 000013
FMT3  R 000023
INPUT R 000026
PRINT R 000043
.MPY  X 000013
.DST  X 000014
.DLD  X 000015
.FMP  X 000016
**   NO ERRORS*
```

```
 PAGE 0002 #01  LINE FORMULI:  DISTANCE, SLOPE, MID-POINT


0002*  PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
0003*  TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
0004*  THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
0005*  POINT (THE MID-POINT) IS CALCULATED.
0006*     DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
0007*  FOUR N-DIGIT REAL NUMBERS AT A TIME.  THE FIRST
0008*  QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
0009*  SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
0010*  THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
0011*  OF THE SECOND POINT.
0012*     THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
0013*  FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
0014*  THAN AN EIGHT DIGIT REAL NUMBER.
0015  00000               NAM LINE
0016  00000 000000  START NOP
0017  00001 026026R       JMP INPUT
0018                      EXT .IOC.,FLOAT,IFIX,SQRT
0019                      EXT .DIO.,.IOI.,.DTA.,.RAR.
0020                      EXT .IOR.,.IAR.
0021  00002 000004R .DATA DEF DATA
0022  00003 000043R .PRIN DEF PRINT
0023  00004 000000  DATA  BSS 4
0024  00010 024106  FMT   ASC 3,(F8.3)
      00011 034056
      00012 031451
0025  00013 024106  FMT2  ASC 8,(F8.3,",",F8.3/)
      00014 034056
      00015 031454
      00016 021054
      00017 021054
      00020 043070
      00021 027063
      00022 027451
0026  00023 024064  FMT3  ASC 3,(4I2)
      00024 044462
      00025 024440
```

```
0028*  INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
0029  00026 000000  INPUT NOP
0030  00027 062131R       LDA =B5
0031  00030 006404        CLB,INB
0032  00031 016005X       JSB .DIO.
0033  00032 000023R       DEF FMT3
0034  00033 000037R       DEF *+4
0035  00034 062132R       LDA =B4
0036  00035 066002R       LDB .DATA
0037  00036 016012X       JSB .IAR.


0039*  THE DISTANCE BETWEEN THE TWO POINTS:
0040                      IFZ
0041  00037 062006R       LDA DATA+2
0042  00040 003004        CMA,INA
0043  00041 042004R       ADA DATA

0045  00042 026047R       JMP *+5
0046                PRINT REP 4
0047  00043 000000        NOP
0047  00044 000000        NOP
0047  00045 000000        NOP
0047  00046 000000        NOP

0049  00047 072043R       STA PRINT
0050                      SUP
0051  00050 016013X       MPY PRINT
0052  00052 072043R       STA PRINT

0054  00053 062007R       LDA DATA+3
0055  00054 003004        CMA,INA
0056  00055 042005R       ADA DATA+1
0057  00056 072044R       STA PRINT+1
0058  00057 016013X       MPY PRINT+1
0059  00061 042043R       ADA PRINT

0061  00062 016002X       JSB FLOAT
0062  00063 016004X       JSB SQRT
0063  00064 016014X       DST PRINT
0064                      XIF


0066*  FIND THE SLOPE OF THE LINE
0067                      IFN
0068                      LDA DATA+2
0069                      CMA,INA
0070                      ADA DATA
0071                      JMP *+5
0072                PRINT REP 4
0073                      NOP
0074                      STA PRINT
0075                      SPC 1
0076                      LDA DATA+3
0077                      CMA,INA
0078                      ADA DATA+1
```

```
 PAGE 0004 #01  LINE FORMULI:  DISTANCE, SLOPE, MID-POINT


0079                       CLB
0080                       DIV PRINT
0081                       DST PRINT
0082                       XIF



0084*  OUTPUT THE RESULT
0085  00066 062133R        LDA =B2
0086  00067 006400         CLB
0087  00070 016005X        JSB .DIO.
0088  00071 000010R        DEF FMT
0089  00072 000076R        DEF *+4
0090  00073 016015X        DLD PRINT
0091  00075 016011X        JSB .IOR.
0092  00076 016007X        JSB .DTA.



0094*  FIND THE MID-POINT OF THE LINE SEGMENT:
0095  00077 062004R        LDA DATA
0096  00100 042006R        ADA DATA+2
0097  00101 006400         CLB
0098  00102 016002X        JSB FLOAT
0099  00103 016016X        FMP =F.5
0100  00105 016014X        DST PRINT

0102  00107 062005R        LDA DATA+1
0103  00110 042007R        ADA DATA+3
0104  00111 006400         CLB
0105  00112 016002X        JSB FLOAT
0106  00113 016016X        FMP =F.5
0107  00115 016014X        DST PRINT+2

0119                       LST



0121                       UNS
0122  00130 026026R        JMP INPUT
      00131 000005
      00132 000004
      00133 000002
      00134 040000
      00135 000000
0123                       END START
**  NO ERRORS*
```

```
 PAGE 0001


0001                   ASMB,R,L,T,N
START R 000000
.IOC. X 000001
FLOAT X 000002
IFIX  X 000003
SQRT  X 000004
.DIO. X 000005
.IOI. X 000006
.DTA. X 000007
.RAR. X 000010
.IOR. X 000011
.IAR. X 000012
.DATA R 000002
.PRIN R 000003
DATA  R 000004
FMT   R 000010
FMT2  R 000013
FMT3  R 000023
INPUT R 000026
PRINT R 000043
.DIV  X 000013
.DST  X 000014
.DLD  X 000015
.FMP  X 000016
**  NO ERRORS*
```

```
PAGE 0002 #01  LINE FORMULI:  DISTANCE, SLOPE, MID-POINT


0002*  PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
0003*  TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
0004*  THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
0005*  POINT (THE MID-POINT) IS CALCULATED.
0006*     DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
0007*  FOUR N-DIGIT REAL NUMBERS AT A TIME.  THE FIRST
0008*  QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
0009*  SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
0010*  THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
0011*  OF THE SECOND POINT.
0012*     THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
0013*  FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
0014*  THAN AN EIGHT DIGIT REAL NUMBER.
0015  00000                NAM LINE
0016  00000 000000  START  NOP
0017  00001 026026R        JMP INPUT
0018                       EXT .IOC.,FLOAT,IFIX,SQRT
0019                       EXT .DIO.,.IOI.,.DTA.,.RAR.
0020                       EXT .IOR.,.IAR.
0021  00002 000004R .DATA  DEF DATA
0022  00003 000043R .PRIN  DEF PRINT
0023  00004 000000  DATA   BSS 4
0024  00010 024106  FMT    ASC 3,(F8.3)
      00011 034056
      00012 031451
0025  00013 024106  FMT2   ASC 8,(F8.3,",",F8.3/)
      00014 034056
      00015 031454
      00016 021054
      00017 021054
      00020 043070
      00021 027063
      00022 027451
0026  00023 024064  FMT3   ASC 3,(4I2)
      00024 044462
      00025 024440
```

```
0028*  INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
0029  00026 000000  INPUT NOP
0030  00027 062123R       LDA =B5
0031  00030 006404        CLB,INB
0032  00031 016005X       JSB .DIO.
0033  00032 000023R       DEF FMT3
0034  00033 000037R       DEF *+4
0035  00034 062124R       LDA =B4
0036  00035 066002R       LDB .DATA
0037  00036 016012X       JSB .IAR.


0039*  THE DISTANCE BETWEEN THE TWO POINTS:
0040                      IFZ
0041                      LDA DATA+2
0042                      CMA,INA
0043                      ADA DATA
0044                      SPC 1
0045                      JMP *+5
0046                PRINT REP 4
0047                      NOP
0048                      SPC 1
0049                      STA PRINT
0050                      SUP
0051                      MPY PRINT
0052                      STA PRINT
0053                      SPC 1
0054                      LDA DATA+3
0055                      CMA,INA
0056                      ADA DATA+1
0057                      STA PRINT+1
0058                      MPY PRINT+1
0059                      ADA PRINT
0060                      SPC 1
0061                      JSB FLOAT
0062                      JSB SQRT
0063                      DST PRINT
0064                      XIF


0066*  FIND THE SLOPE OF THE LINE
0067                      IFN
0068  00037 062006R       LDA DATA+2
0069  00040 003004        CMA,INA
0070  00041 042004R       ADA DATA
0071  00042 026047R       JMP *+5
0072                PRINT REP 4
0073  00043 000000        NOP
0073  00044 000000        NOP
0073  00045 000000        NOP
0073  00046 000000        NOP
0074  00047 072043R       STA PRINT

0076  00050 062007R       LDA DATA+3
0077  00051 003004        CMA,INA
0078  00052 042005R       ADA DATA+1
```

```
PAGE 0003 #01  LINE FORMULI:  DISTANCE, SLOPE, MID-POINT


0028*  INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
0029  00026 000000  INPUT NOP
0030  00027 062123R       LDA =B5
0031  00030 006404        CLB,INB
0032  00031 016005X       JSB .DIO.
0033  00032 000023R       DEF FMT3
0034  00033 000037R       DEF *+4
0035  00034 062124R       LDA =B4
0036  00035 066002R       LDB .DATA
0037  00036 016012X       JSB .IAR.


0039*  THE DISTANCE BETWEEN THE TWO POINTS:
0040                      IFZ
0041                      LDA DATA+2
0042                      CMA,INA
0043                      ADA DATA
0044                      SPC 1
0045                      JMP *+5
0046                PRINT REP 4
0047                      NOP
0048                      SPC 1
0049                      STA PRINT
0050                      SUP
0051                      MPY PRINT
0052                      STA PRINT
0053                      SPC 1
0054                      LDA DATA+3
0055                      CMA,INA
0056                      ADA DATA+1
0057                      STA PRINT+1
0058                      MPY PRINT+1
0059                      ADA PRINT
0060                      SPC 1
0061                      JSB FLOAT
0062                      JSB SQRT
0063                      DST PRINT
0064                      XIF


0066*  FIND THE SLOPE OF THE LINE
0067                      IFN
0068  00037 062006R       LDA DATA+2
0069  00040 003004        CMA,INA
0070  00041 042004R       ADA DATA
0071  00042 026047R       JMP *+5
0072                PRINT REP 4
0073  00043 000000        NOP
0073  00044 000000        NOP
0073  00045 000000        NOP
0073  00046 000000        NOP
0074  00047 072043R       STA PRINT

0076  00050 062007R       LDA DATA+3
0077  00051 003004        CMA,INA
0078  00052 042005R       ADA DATA+1
```

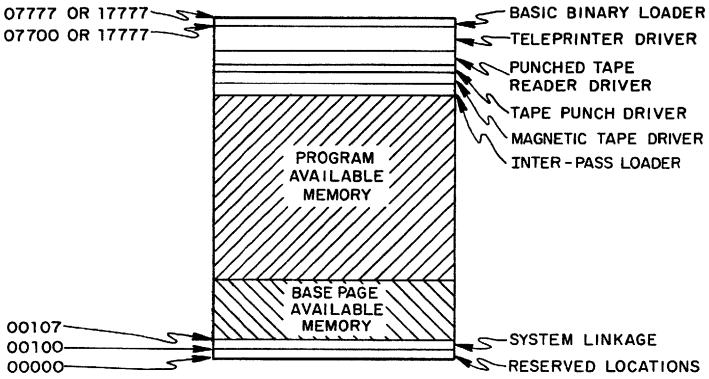
SYSTEM INPUT/OUTPUT SUBROUTINES E

The System Input/Outout (SIO) subroutines may be used to perform basic input/output operations for programs in absolute form. †

MEMORY ALLOCATION

These drivers are stored in high memory immediately preceding the Basic Binary Loader. The Teleprinter driver must be loaded first; it is stored in the highest portion of this area. The drivers for the Punched Tape Reader (or Marked Card Reader), the Tape Punch, and the Magnetic Tape Unit may then be loaded. The sequence of loading must fall within this order, depending on your equipment configuration: Line Printer Driver, Punched Tape Reader Driver (or Marked Card Reader), Tape Punch Driver, Magnetic Tape Driver, and if needed, the MTS Boot.

The drivers are accessed through 15-bit absolute addresses which are stored in the System Linkage area starting at location 101_8. The allocation of memory is as follows:



† The SIO subroutines are designed for use with FORTRAN, Assembler, Symbolic Editor, etc.; however, they may be used with any absolute object program.

OPERATION AND CALLING SEQUENCE:

PAPER TAPE DEVICES

All data transmission is accomplished without interrupt control, and therefore, operations are not buffered by the drivers. Control is not returned to the calling program until an operation is completed. Data is transferred to and from buffer storage areas specified in the user program.

The general form of the paper tape input/output calling sequence is:

```
LDA <buffer length> (words or characters)

LDB <buffer address>

JSB 10fB,I     (f is Input/Output function)

<normal return>
```

Register Contents

When the JSB is performed, the A-Register must contain the length of the buffer storage area and the B-Register, the address of the buffer. Control returns to the location following the JSB. After an input request is completed, the A-Register contains a positive integer indicating the number of characters, or a negative integer to indicate the words transmitted or zeros, if an End-of-Tape (EOT) condition occurred.

The digit supplied for f in the JSB instruction determines the paper tape input/output function to be performed. The value of the operand address is the location in the System Linkage that contains the absolute address of the driver entry point. The following are available:

101	Input
102	List Output
103	Punch Output
104	Keyboard Input—ASCII data is read from Teleprinter and printed as it is received.

If the Teleprinter driver alone is loaded, these locations point to entry points of this driver. If Punched Tape Reader and Tape Punch drivers are in memory, location 101 points to the Punched Tape Reader driver and location 103, to the Tape Punch driver. If the latter are to be used, they must be loaded after the Teleprinter driver.

OPERATION AND CALLING SEQUENCE: MAGNETIC TAPE DRIVER

As with the Paper Tape SIO drivers, all data transmission is accomplished without interrupt control. Control is not returned to the calling program until an operation is completed. (Rewind and rewind standby are the only exceptions to this. In these cases return is made as soon as the command is accepted.)

The general form of the calling sequence is:

```
LDA  <buffer length>   or  <file count>
LDB  <buffer address>  or  <record count>
JSB   107B,I
OCT  <command code>
< EOF/EOT/SOT return>
<error return>
<normal return>
```

NOTE: Location 107_8 must contain the address of the magnetic tape driver.

Register Contents

Before initiating read or write operations, the A-Register must contain the buffer length. This will be a positive integer if length is defined in characters and a negative integer if length is defined in words. The B-Register must contain the buffer address.

Before initiating tape positioning operations, the A-Register must contain the number of files that are to be spaced. A positive integer indicates forward spacing; a negative integer indicates backward spacing. The B-Register contains the number of records that are to be spaced. A positive integer indicates forward spacing; a negative integer indicates backward spacing. The positioning may be defined in terms of any combination of forward or backward spacing of files and records (e.g., space forward two files then backspace three records). If files only or records only are to be spaced, the contents of the other register should be zeros.

The registers are not used when entering the subroutine to perform one of the following operations:

Write end-of-file
Write file gap
Rewind
Rewind/Standby
Status

Linkage Address

107B is the System Linkage word that contains the absolute address of the entry point for the Magnetic Tape driver.

On return from a read operation, the A-Register contains a positive value indicating the number of words or characters transmitted.

On return from all operations except Rewind and Rewind/ Standby the B-Register contains status of the operation (See Status).

MAGNETIC TAPE OPERATIONS

The magnetic tape driver will perform the following operations. The pertinent operation is specified by the command code which appears after the OCT in the calling sequence.

Operation	Command Code
Read	0
Write	1
Write End-of-File	2
Rewind (Auto mode)	3
Position	4
Rewind/Standby (Local mode)	5
Gap	6
Status	7

Read

One tape record is read into the buffer. The number of characters or words read is stored in the A-Register. The value will be equal to the buffer length except when the data on tape is less than the length of the buffer. One tape record is read to transfer the number of characters specified into the buffer. The number of characters in that record (not the number transferred) will be stored in the A-Register. If the tape record exceeds the buffer length, the data will be read into the buffer until the buffer is filled, the remainder of the record will be skipped. If the length of an input buffer is an odd number of characters, a read operation will result in the overlaying of the character following the last character of the buffer; the subroutine actually transmits full words only.

Three attempts are made to read the record before returning control to the parity error address.

If an EOT condition exists at the time of entry, the command will be ignored and control will be returned to the EOT/EOF address.

If the buffer length specified is 0 control will return to the normal address without any tape movement.

The input buffer storage area can be as large or as small as needed. The number of characters in the tape record will be stored in the A-Register.

Write

The contents of the buffer is written on tape preceded by the record length. Since a minimum of 7 tape characters (12 on 3030) may be written, short records are padded.

If the end-of-tape is detected during the write operation, the normal return is used. The next write operation, however, results in a return of control of the EOF/EOT location; no data is written. If an EOT condition exists at the time of entry, the command will be ignored and control will be returned to the EOT/EOF address.

Write End-of-File

A standard EOF character (17_8 for 2020, 23_8 for 3030) is written on tape. Control returns to the normal location with the EOF status on the B-Register. No gap is written.

If the end of tape was reached on a previous write command, control returns to the EOF/EOT location; the character is written.

Rewind

This command initiates a rewind operation and then immediately returns control to the normal location.

The calling sequence for a Rewind operation consists of:

```
JSB       107B,I
OCT       3
<normal return>
```

The user need not test status on the rewind operation before issuing the next call.

Position

This is the general command to move the tape. Both file and record operations may be defined in the same operation. Either may be specified for forward or backward spacing. At the completion of the operation the tape will be positioned ready for reading or writing.

An attempt to space beyond the End-of-Tape or Start-of-Tape will terminate the positioning operation and return control to the EOF/EOT/SOT location.

Rewind/Standby

This causes the tape to be positioned at load point and switches the device to local status. Control returns to the normal location immediately after the operation is initiated.

The calling sequence for a Rewind / Standby operation consists of:

```
JSB      107B,I
OCT      5
<normal return>
```

An attempt to issue another call on this device results in a halt (102044). The device must be switched to AUTO before the program can continue.

Gap

This command causes a 3-inch gap to be written on the tape.

If the End-of-Tape was reached on a previous write command, control returns to the EOF/EOT location; the gap is not written.

Status

This command returns certain status bits in the B-Register. The driver performs a clear command whenever it is entered and as a result the only bits that are valid indicators are:

Start-of-Tape
End-of-Tape
Write Not Enabled

All other commands (except Rewind and Rewind/Standby) provide valid status replies on return to the program.

The status reply consists only of bits 8-0 and has the following significance:

Bits 8-0	Condition
1xxxxxxxx	Local - The device is in local status
x1xxxxxxx	EOF- An End-of-File character (17_8 for 7 track, 23_8 for 9) has been detected while reading, forward spacing, or backspacing.
xx1xxxxxx	SOT - The Start-of-Tape marker is under the photo sense head.
xxx1xxxxx	EOT - The End-of-Tape reflective marker is sensed while the tape is moving forward. The bit remains set until a rewind command is given.
xxxx1xxxx	Timing - A character was lost.
xxxxx1xxx	Reject - a) Tape motion is required and the unit is busy. b) Backward tape motion is required and the tape is at load point. c) A write command is given and the tape reel does not have a write enable ring.
xxxxxx1xx	Write not enabled - Tape reel does not have write enable ring or tape unit is rewinding.
xxxxxxx1x	Parity error - A vertical or longitudinal parity error occurred during reading or writing. (Parity is not checked during forward or backward spacing operations.)
xxxxxxxx1	Busy - The tape is in motion or the device is in local status.

Following is a table summarizing the tape commands:

Operation	Command Code	Call		Return	
		A	B	A	B
Read	Ø	Buffer Length	Buffer Address	Buffer or Record Length	Status
Write	1	Buffer Length	Buffer Address	Buffer Length	Status
Write EOF	2	-	-	-	Status
Rewind (Auto mode)	3	-	-	-	-
Position	4	Number of Files, Direction	Number of Records, Direction	-	Status
Rewind/ Standby (Local mode)	5	-	-	-	-
Gap	6	-	-	-	Status
Status	7	-	-		Status

Additional Linkage Addresses

Other locations in the System Linkage area contain the following:

100_8 Used by the standard software system to store a JMP to the transfer address.

105_8 First word address of available memory.

106_8 Last word address of available memory.

The latter two locations may be accessed by an absolute program. The user may store the first word of available memory in 105 by performing the following:

```
ORG  105B
ABS  < last location of user program +1 >
```

The last word of available memory is established by the drivers; it is the location immediately preceding the first location used by the last driver loaded.

BUFFER STORAGE AREA

The Buffer Address is the location of the first word of data to be written on an output device or the first word of a block reserved for storage of data read from an input device. The length of the buffer area is specified in the A-Register in terms of ASCII input or output characters or binary output words. For binary input, the length of the buffer is the length of the record which is specified in the first character of the record. ASCII and binary input record lengths are given as positive integers. The length of a binary output record is specified as the two's complement of the number of words in the record.

In addition to describing the buffer area in the calling sequence, (or first word of binary input record), the area must also be specifically defined in the program, for example with a BSS instruction.

Record Formats

ASCII Records (Paper Tape)

An ASCII record is a group of characters terminated by an end-of-record mark which consists of a carriage return, CR, and a line feed, LF.

For an input operation, the length of the record transmitted to the buffer is the number of characters designated in the A-Register, or less if an end-of-record mark is encountered before the character count is exhausted. The codes for CR and LF are not transmitted to the buffer. An end-of-record mark preceding the first data character is ignored.

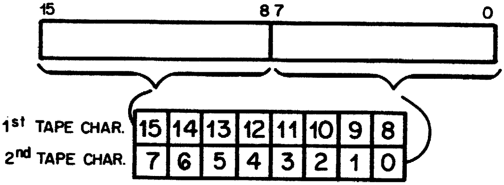
For an output operation, the length of the record is determined by the number of characters designated in the request. An end-of-record mark is supplied at the end of each output operation by the driver.

If a RUB OUT code followed by a CR LF is encountered on input from the Teleprinter or Punched Tape Reader, the current record is ignored (deleted) and the next record transmitted. †

If less than ten feed frames (all zeros) are encountered before the first data character from the Punched Tape Reader, they are ignored. Ten feed frames are interpreted as an end-of-tape condition.

Binary Records (Paper Tape)

A binary record is transmitted exactly as it appears in memory or on 8-level paper tape. Each computer word is translated into two tape "characters" (and vice versa) as follows:



For an output operation, the record length is the number of words designated by the value in the A-Register (the value is the two's complement of the number of words). For input operations, the first word of the record contains a positive integer in bits 15-8 specifying the length (in words) of the record including the first word.

† RUB OUT which appears on the Teleprinter keyboard is synonymous with the ASCII symbol DEL

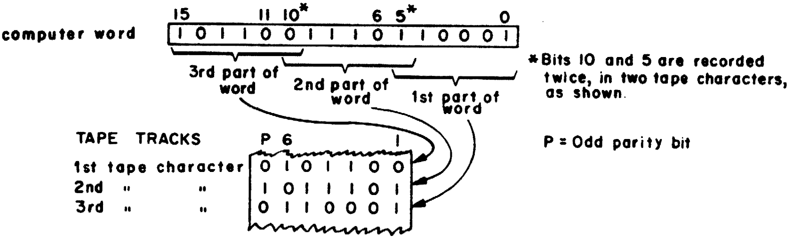
On input operations if less than ten feed frames precede the first data character, they are ignored; ten feedframes are interpreted as an end-of-tape condition. On output, the driver writes four feed frames to serve as a physical record separator.

Binary Records (Magnetic Tape)

The Magnetic Tape subroutine reads and writes binary (odd parity) records only. A record count is supplied by the driver as the first word of the record. This allows automatic padding of short records to the minimum record length with automatic removal of the padded portion of the record on read.

2020 7-LEVEL TAPE

Each Computer word is translated into three tape "characters" (and vice versa) as follows:

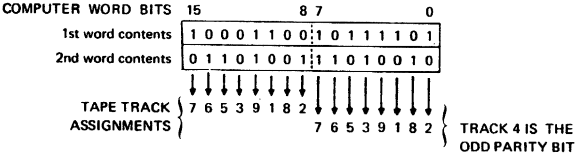


*Bits 10 and 5 are recorded twice, in two tape characters, as shown.

P = Odd parity bit

3030 9-LEVEL TAPE

Each Computer Word is translated into Two tape "characters" by repositioning the bits in the following scheme:



TRACK 4 IS THE ODD PARITY BIT

TAPE TRACKS	9					4			1
1st tape character	1	0	1	0	0	0	0	0	1
2nd tape character	1	0	1	0	1	1	1	1	1
3rd tape character	1	0	0	1	1	1	0	1	0
4th tape character	0	1	1	1	0	1	1	0	0

OPERATING AND CALLING SEQUENCE: MARK SENSE CARD READER

The SIO Mark Sense Card Reader Driver overlays the Punched Tape Reader Driver exactly, therefore, only one or the other of these two drivers may be used in any one SIO System configuration. Further, the driver has no binary read capability; if this ability is needed, the BCS Mark Sense Card Reader Driver will have to be used.

All data transmission is accomplished without interrupt control. Execution control is not returned to the calling program until a complete card has been read.

The general form of the calling sequence is:

```
LDA    < character count >    (positive)
LDB    < buffer address >
JSB    < 101B, I >
<normal return >
```

Register Contents

Before the JSB is executed, the A-Register must contain the character count (the buffer length) and the B-Register must contain the buffer address. Control returns to the location following the JSB; then the A-Register will contain the number of characters transmitted not including trailing blanks, or, if a transmission error was detected, it will contain all zeros.

CALLING SEQUENCES

The Formatter is a library subroutine used by FORTRAN and ALGOL to input or output data. An assembler program may access the Formatter routine with a 5 to 9 line calling sequence depending on the form of the call.

I. Format Definition

```
                INPUT                         OUTPUT

                LDA     (unit)                LDA    (unit)
Formatted       CLB,INB                       CLB
                JSB     .DIO.                 JSB    .DIO.
                DEF     (fmt) or ABS 0        DEF    (fmt)
                DEF     (end of list)         DEF    (end of list)

                LDA     (unit)                LDA    (unit)
Binary          CLB,INB                       CLB
                JSB     .BIO.                 JSB    .BIO.
```

where

unit — refers to the unit reference number of the device to be called

fmt — is the label of an ASC pseudo instruction which defines the format specification

end of list — is the location immediately following the last parameter of the calling sequence; it is to this location that the Formatter returns control.

ABS 0 — is an option for free field input

formatted input/output — is in ASCII code

binary input/output — is in binary code

II. Element Definition

```
                    INPUT                        OUTPUT

Real Variable       JSB     .IOR.                DLD     x
                    DST     x                    JSB     .IOR.

Integer Variable    JSB     .IOI                 LDA     i
                    STA     i                    JSB     .IOI.

Array               LDA     array length
                    LDB     array address
                    JSB     .RAR. (real)  or  .IAR. (integer)
```

where

x or i — are addresses, real or integer, of the data

array length — is the number of elements (not the number of memory locations) in the block of data. (Maximum length is equivalent to 60 computer words.)

III. Terminator

```
                    INPUT                   OUTPUT

                    (none)                  JSB     .DTA.
```

Symbols such as .DIO., .IOR., etc., are entry points to the Formatter; all entry points used in the calling sequence must be declared external with an EXT pseudo code.

Data stored in memory may be converted internally from one format to another with the following initial call.

```
        LDA         =BO
        JSB         .DIO.
        DEF         buffer
        DEF         (fmt)
        DEF         (end of list)
          .
          .
        Element Definition
          .
        Terminator
```

where buffer is the address of the data to be converted.

FORMAT SPECIFICATIONS

Below are listed the format conversion and editing specifications.

rAw	Alphanumeric character
rEw.d	Real number with exponent
rFw.d	Real number without exponent
rIw	Decimal integer
r@w rKw	Octal integer
nX	Blank field descriptor
nHh$_1$. . . h$_n$ r‘‘h$_1$. . . h$_n$’’	Heading and labeling descriptors
r/	Begin new resord

where

r is the number of times the entire format is repeated

w is the number of digits in the format

d is the number of digits to the right of the decimal point (w-d should be greater than or equal to 4)

n is the number of characters or spaces

h's represents the ASCII characters

Aw translates alphanumeric data to or from memory. If w is greater than 2 only the last two characters are processed; if w is 1, the single character is read into or written from the right-half of the computer word.

Ew converts data to a real number. On output, data may consist of integer, fraction, and exponent subfields.

$$\frac{+}{N}\ n\ .\ .\ .\ n.n\ .\ .\ .\ n\ \frac{+}{E}\ ee$$

On output, data appears in floating point form.

$$\underline{\triangle}.\ x_1\ .\ .\ .\ x_d\ E \pm ee$$

Fw — For output operations real numbers in memory are converted to character form which will appear right justified in decimal form. Input is identical to the E specification input.

$\underline{\wedge}\, x \ldots x . x \ldots x$

Iw — translates decimal integers toor from memory

$\underline{\wedge}\, x_1 \ldots x_d$

@w and Kw — translates octal integers to or from memory.

$\underline{\wedge}\, x_1 \ldots x_d$

nH$h_1 \ldots h_n$ — provides for the transfer of any combination of 8-bit ASCII characters, including blanks.

r"$h_1 \ldots h_n$" — also transfers ASCII characters; field length is not specified, quotation marks are not transferred.

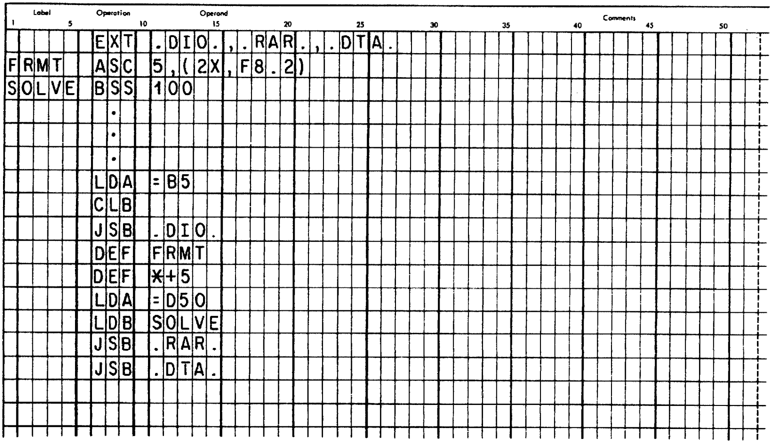
(For a more detailed description of the Format specifications see the FORTRAN Programmer's Reference Manual, Section 7.)

EXAMPLE

Below is an example of a calling sequence to the Formatter that will output the contents of a block data, SOLVE, such that each number is printed on the teleprinter in the following manner:

xxxxxx.xx

SOLVE occupies 100_{10} memory locations; the data stored there is in floating point form.


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      EXT .DIO.,.RAR.,.DTA.
FRMT  ASC 5,(2X,F8.2)
SOLVE BSS 100
       .
       .
       .
      LDA =B5
      CLB
      JSB .DIO.
      DEF FRMT
      DEF *+5
      LDA =D50
      LDB SOLVE
      JSB .RAR.
      JSB .DTA.
```

ASSEMBLER ERROR MESSAGES

During the compilation or assembly of programs, error messages are typed on the list output device to aid the programmer in debugging programs. Errors detected in the source program are indicated by a 1- or 2- letter mnemonic followed by the sequence number and the first 62 characters of the statement in error. The messages are printed on the output device during the passes indicated.

For Extended Assembler, error listings produced during Pass 1 are preceded by a number which identifies the source input file where the error was found. Pass 2 and 3 error messages are preceded by a reference to the previous page of the listing where an error message was written. The first error will refer to page "O".

Error Code	Pass	Description
CS	1	Control statement error: a) The control statement contained a parameter other than the legal set. b) Neither A nor R, or both A and R were specified. c) There was no output parameter (B, T, or L.)
DD	1	Doubly defined symbol: A name defined in the symbol table appears more than once as: a) A label of a machine instruction. b) A label of one of the pseudo operations: BSS EQU ASC ABS DEC OCT DEF Arithmetic subroutine call DEX

Error Code	Pass	Description
		c) A name in the Operand field of a COM or EXT statement. d) A label in an instruction following a REP pseudo operation. e) Any combination of the above.
		An arithmetic subroutine call symbol appears in a program both as a pseudo instruction and as a label.
EN	1	The symbol specified in an ENT statement has already been defined in an EXT or COM statement.
ENØØØ <symbol>	2	The entry point specified in an ENT statement does not appear in the label field of a machine or BSS instruction. The entry point has been defined in the Operand field of an EXT or COM statement, or has been equated to an absolute value.
IF	1	An IFZ or an IFN follows either an IFZ or an IFN without an intervening XIF. The second pseudo instruction is ignored.
IL	1	Illegal instruction: a) Instruction mnemonic cannot be used with type of assembly requested in control statement. The following are illegal in an absolute assembly: NAM EXT ENT COM ORB Arithmetic subroutine calls b) The ASMB statement has an R parameter, and NAM has been detected after the first valid Opcode.
IL	2 or 3	Illegal character: A numeric term used in the Operand field contains an illegal character (e.g. an octal constant contains other than +, –, or Ø–7).
		Illegal instruction: ORB in an absolute assembly

Error Code	Pass	Description
M	1, 2 or 3	Illegal operand: a) Operand is missing for an Opcode requiring one. b) Operands are optional and omitted but comments are included for: END HLT c) An absolute expression in one of the following instructions from a relocatable program is greater than 77_8. Memory Reference DEF Arithmetic subroutine calls d) A negative operand is used with an Opcode field other than ABS, DEX, and OCT. e) A character other than I follows a comma in one of the following statements: ISZ ADA AND DEF JMP ADB XOR Arithmetic subroutine calls JSB LDA IOR LDB CPA STA CPB STB f) A character other than C follows a comma in one of the following statements: STC MIB CLC OTA LIA OTB LIB HLT MIA

Error Code	Pass	Description
		g) A relocatable expression in the operand field of one of the following: ABS ASR RRL REP ASL LSR SPC RRR LSL
		h) An illegal operator appears in an Operand field (e.g. + or - as the last character).
		i) An ORG statement appearing in a relocatable program includes an expression that is base page or common relocatable or absolute.
		j) A relocatable expression contains a mixture of program, base page, and common relocatable terms.
		k) An external symbol appears in an operand expression or is followed by a common and the letter I.
		l) The literal or type of literal is illegal for the operation code used (e.g., STA = B7).
		m) An illegal literal code has been used (e.g., LDA = Ø77).
		n) An integer expression in one of the following instructions does not meet the condition $1 \leqslant n \leqslant 16$. The integer is evaluated modulo 2^4. ASR RRR LSR ASL RRL LSL
		o) The value of an 'L' type literal is relocatable.

Error Code	Pass	Description
NO	1, 2, 3	No origin definition: The first statement in the assembly containing a valid opcode following the ASMB control statement (and remarks and/or HED, if present) is neither an ORG nor a NAM statement. If the A parameter was given on the ASMB statement, the program is assembled starting at 2ØØØ; if an R parameter was given, the program is assembled starting at zero.
OP	1, 2, 3	Illegal Opcode preceding first valid Opcode. The statement being processed does not contain an asterisk in position one. The statement is assumed to contain an illegal Opcode; it is treated as a remarks statement.
OP	1,2, or 3	Illegal Opcode: A mnemonic appears in the Opcode field which is not valid for the hardware configuration or assembler being used. A word is generated in the object program.
OV	1,2 or 3	Numeric operand overflow: The numeric value of a term or expression has overflowed its limit: $1 \geqslant N \geqslant 16$ Shift-Rotate Set $2^{6}-1$ Input/Output, Overflow, Halt $2^{10}-1$ Memory Reference (in absolute assembly) $2^{15}-1$ DEF and ABS operands; data generated by DEC; or DEX: expressions concerned with program location counter. $2^{16}-1$ OCT
R?	Before 1	An attempt is made to assemble a relocatable program following the assembly of an absolute program.

Error Code	Pass	Description
SO		There are more symbols defined in the program than the symbol table can handle.
SY	1, 2, 3	Illegal Symbol: A Label field contains an illegal character or is greater than 5 characters. A label with illegal characters may result in an erroneous assembly if not corrected. A long label is truncated on the right to 5 characters.
SY	2 or 3	Illegal Symbol: A symbolic term in the Operand field is greater than five characters; the symbol is truncated on the right to 5 characters. Too many control statements: A control statement has been input both on the teleprinter and the source tape or the source tape contains more than one control statement. The Assembler assumes that the source tape control statement is a label, since it begins in column 1. Thus, the commas are considered as illegal characters and the "label" is too long. The binary object tape is not affected by this error, and the control statement entered via the teleprinter is the one used by the Assembler.
TP	1,2, or 3	An error has occurred while reading magnetic tape.
UN	1,2, or 3	Undefined Symbol: a) A symbolic term in an Operand field is not defined in the Label field of an Instruction or is not defined in the Operand field of a COM or EXT statement.

Error Code	Pass	Description
UN	1,2, or 3	Undefined Symbol: (continued) b) A symbol appearing in the Operand field of one of the following pseudo operations was not defined previously in the source program: BSS ASC EQU ORG END

CONSOLIDATED CODING SHEET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	AND	001		0	Z/C						← Memory Address →				
D/I	XOR	010		0	Z/C										
D/I	IOR	011		0	Z/C										
D/I	JSB	001		1	Z/C										
D/I	JMP	010		1	Z/C										
D/I	ISZ	011		1	Z/C										
D/I	AD*	100		A/B	Z/C										
D/I	CP*	101		A/B	Z/C										
D/I	LD*	110		A/B	Z/C										
D/I	ST*	111		A/B	Z/C										

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SRG	000		A/B	0	D/E	*LS		000	CLE	D/E	SL*	*LS		000
				↓	↓	↓	*RS		001		↓		*RS		001
							R*L		010				R*L		010
							R*R		011				R*R		011
							*LR		100				*LR		100
							ER*		101				ER*		101
							EL*		110				EL*		110
							*LF		111				*LF		111
				NOP	000				000		000				000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ASG	000		A/B	1	CL*	01	CLE	01	SEZ	SS*	SL*	IN*	SZ*	RSS
				↓	↓	CM*	10	CME	10						
						CC*	11	CCE	11						

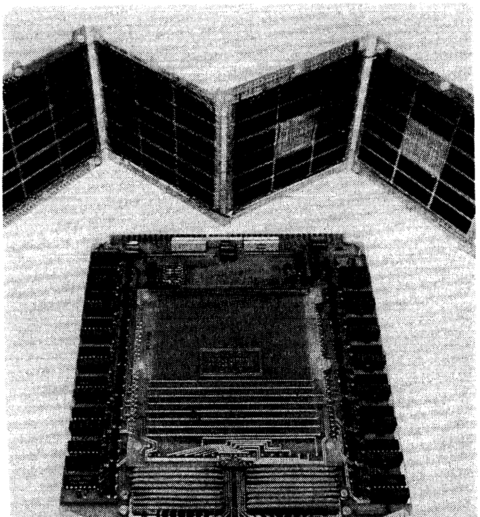
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IOG	000			1	H/C	HLT		000			← Select Code →			
↓					1	0	STF		001						
					1	1	CLF		001						
					1	0	SFC		010						
					1	0	SFS		011						
				A/B	1	H/C	MI*		100						
				A/B	1	H/C	LI*		101						
				A/B	1	H/C	OT*		110						
				0	1	H/C	STC		111						
				1	1	H/C	CLC		111						
					1	0	STO		001		000			001	
					1	1	CLO		001		000			001	
					1	H/C	SOC		010		000			001	
					1	H/C	SOS		011		000			001	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	EAU	000		MPY**		000		010			000			000	
↓				DIV**		000		100			000			000	
				DLD**		100		010			000			000	
				DST**		100		100			000			000	
				ASR		001		000		0	1				
				ASL		000		000		0	1				
				LSR		001		000		1	0			← number of bits →	
				LSL		000		000		1	0				
				RRR		001		001		0	0				
				RRL		000		001		0	0				

Notes: * = A or B.
D/I, A/B, Z/C, D/E, H/C coded: 0/1.
**Second word is Memory Address.

Basic Control System Reference Manual

A 2100A 8K corestack with sense amplifiers and diode decoding matrix is contained on a single plug-in circuit board.



CONTENTS

GENERAL DESCRIPTION 1

The Basic Control System (BCS) provides an efficient loading and input/output control capability for relocatable programs produced by the HP Assembler, HP FORTRAN, FORTRAN IV, or HP ALGOL. BCS is modular in design and is constructed to fit each user's hardware configuration.

The Basic Control System performs the following functions:

- Loads and links relocatable programs
- Creates indirect and base page addressing when necessary
- Selects and loads referenced library routines
- Processes I/O requests and services I/O interrupts

The Basic Control System is comprised of two distinct parts: input/output subroutines and the Relocating Loader. Associated with the Basic Control System are two other systems: Prepare Control System and the Debugging System.

The Relocating Loader loads and links relocatable object programs generated by the Assembler, FORTRAN, and ALGOL. It also links the object programs with the input/output subroutines and any library subroutines referred to in the programs. The Prepare Control System is used to adapt the Basic Control System program to a particular hardware configuration. The Debugging System is a relocatable program that BCS loads after the object program(s); with the debugging program the programmer can find errors in his program.

The minimum equipment configuration required for the Basic Control System (and Prepare Control System) is as follows:

2100 family computer with 4K memory

Teleprinter

1.1 INPUT/OUTPUT SUBROUTINES

The input/output package consists of an Input/Output Control subroutine and driver subroutines for the peripheral devices. Input/output operations are specified as symbolic calling sequences in Assembler language. These requests are translated into object code calls to the I/O Control subroutine. The subroutine interprets the call and directs the request to the proper driver. The driver initiates the operation and returns control to the calling program. Whenever interrupt occurs, the driver temporarily resumes control to transfer the next element of data. When the operation is completed, the I/O Control subroutine makes the status of the operation available for checking by the program.

The input/output package allows device independent programming; a device is specified in terms of a unit-reference number rather than a channel number or select code. Furthermore, the user need not be concerned about how data is transmitted (by bit, by character, etc.), he need only specify the number of words or characters and the location in memory where the data is stored.

1.2 RELOCATING LOADER

The Relocating Loader loads object code programs produced by the Assembler, FORTRAN and ALGOL. The linking capability of the Loader allows the user to divide a program into several subprograms, to assemble and test each separately, and finally to execute all as one program. Object subprograms produced by the Assembler may be combined with object subprograms produced by FORTRAN and ALGOL. The subprograms are linked through symbolic entry points and external references.

The loader also provides indirect addressing whenever an operand of an instruction does not fall in the same page as that into which the instruction is being loaded. This allows a program to be designed without concern for page boundaries.

An optional feature of the loader allows the user to obtain an absolute dump of a relocatable program plus the Basic Control System and those library subroutines that were referenced by the program. The process of generating the absolute program is such that instructions (not just common storage) may be allocated to the area normally occupied by the loader. This feature may also be utilized for a program which has reached "production" status; absolute format requires less loading time because an absolute program is loaded by the Basic Binary Loader.

The following information is relevant to the Relocating Loader used in core memory greater than 4K:

a. When the Relocating Loader is not requested to produce an absolute version of a program, it sets all unused locations in memory to 106055_8 (a unique halt instruction) so that a halt will occur if any should be executed. This is useful for detecting errors in programs.

b. A certain portion of the BCS Relocating Loader must always be resident in core while the BCS is in use. This portion of the Relocating Loader contains a segment labeled HALT, which is used by the new version of the .STOP routine in the Relocatable Library. The final halt instruction for the BCS is directly associated with this entry point for use in one of two ways. The final halt instruction remains unchanged if paper tape operation is used, but it is changed to JSB 00106_8, I (a call to the Inter-Pass Loader of the Magnetic Tape System) if the BCS is run using MTS.

For further information on the BCS and its relation to the Magnetic Tape System see the Magnetic Tape System manual, HP 02116-91752.

1.3 PREPARE CONTROL SYSTEM

Prepare Control System is a special purpose program which produces an absolute version of the Basic Control System from relocatable BCS subprograms. During the construction of the absolute BCS, the user also establishes the relationships among I/O channel numbers, drivers, interrupt entry points in the drivers, and unit-reference numbers. Prepare Control System is used when the configuration of the hardware is defined initially or whenever there is a modification or expansion to the configuration.

1.4 DEBUGGING SYSTEM

The debugging routine provides aids in program testing. Options provided by the routine will print selected areas of memory, trace portions of the program during execution, modify the contents of selected areas in memory, modify simulated computer registers, halt execution of the program at specified breakpoints, and initiate execution at any point in the program.

The Basic Control System provides the facility to request input/output operations in the form of five-word calling sequences in assembly language. The Basic Control System interprets the call, initiates the operation, and returns control to the calling program. When the data transfer is complete, the System provides status information which may be checked by the program. Interrupts which occur during or on termination of the transfer are processed entirely by the System; interrupt handling subroutines are not required in the user's program.

2.1 GENERAL CALLING SEQUENCE

The general form of the input/output request is:

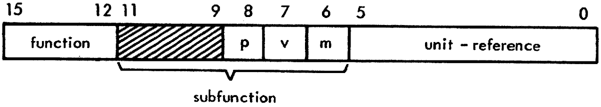
```
     EXT        .IOC.
      .
      .
      .
     JSB        .IOC.

     OCT        <function> <subfunction> <unit-reference>

    {JSB}
    {JMP}       reject address  <error return>

     DEF        buffer address

    {DEC}
    {OCT}       buffer length

     <normal return>
```

2.1.1 INPUT/OUTPUT SUBROUTINE (.IOC.)

.IOC. is the symbolic entry point name of the input/output control subroutine within the Basic Control System. All input/output operations are requested by performing a jump subroutine (JSB) to this entry point. The input/output control subroutine returns control to the calling program at the first location following the last word of the calling sequence. Programs referring to .IOC. must declare it as an external symbol.

2.1.2 FUNCTION, SUBFUNCTION, AND UNIT-REFERENCE

The second word of the request determines the function to be performed and the unit of equipment for which the action is to be taken. In assembly language, this information may be supplied in the form of an octal constant. The bit combinations that comprise the constant are as follows:



Function

The function (bits 15-12) indicates the basic read/write operation:

Function Name	Code (octal)
Read	01
Write	02

Subfunction

The subfunction (bits 11-6) defines the options for certain read/write operations:

p = 1 Print input: The ASCII data read from the Teleprinter is printed as it is received.

v = 1 Variable length binary input: The value in bits 15-8 of the first word on an input paper tape indicates the length of the record (including the first word). If the value exceeds the length of the buffer, only the number of words specified as the buffer length are read. If v = 0, the buffer length field always determines the length of record to be transmitted. If the device does not read paper tape, the parameter is ignored.

m = 1 Mode: The data is transmitted in binary form exactly as it appears in memory or on the external device. If m = 0, the data is transmitted in ASCII or BCD format.

Unit-Reference

The value specified for the unit-reference field indicates the unit of equipment on which the operation is to be performed. The number may represent a standard unit assignment or an installation unit assignment. Standard unit numbers are as follows:

Number	Name	Usual Equipment Type
1	Keyboard Input	Teleprinter
2	Teleprinter Output	Teleprinter
3	Program Library	Punched Tape Reader
4	Punch Output	Tape Punch
5	Input	Punched Tape Reader
6	List Output	Teleprinter

Installation unit numbers may be in the range 7_8-74_8 with the largest value being determined by the number of units of equipment available at the installation. The particular physical unit that is referenced depends on the manner in which equipment is defined within the Basic Control System by the installation. When the Basic Control System configuration is established, an equipment table (EQT) is created. This table defines the type of equipment (Teleprinter, magnetic tape, etc.), the channel on which each unit is connected, and other related details. The ordinal of the unit's entry in this table is the value specified as a unit-reference number for an installation unit. Since numbers 1-6 are reserved as standard unit numbers, the first unit described in the table is referred to by the number 7_8; the second, 10_8; the third, 11_8; and so forth. The entries for one possible equipment table configuration might establish the following relationships:

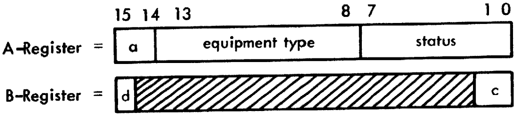
Installation unit number (ordinal)	Device	I/O Channel
7	Teleprinter	12 or 12 and 13
10	Punched Tape Reader	10
11	Tape Punch	11

The standard unit numbers are associated with physical equipment via a standard equipment table (SQT) and EQT. The SQT is a list of references to the EQT. SQT is also created by the installation when the BCS configuration is established. Each standard unit may be a separate device, or a single device may be accessed by several standard unit numbers as well as an installation unit number. (For complete details on the SQT and EQT, see Appendices B and C.)

2.1.3 REJECT ADDRESS

The content of the third word of the calling sequence is normally a JSB or a JMP to a reject address which is the start of a user subroutine designed to determine the cause of a reject and take appropriate action.

The Basic Control System transfers control to this address if the input/output operation can not be performed. On transfer, the system provides status information that may be checked by the user's program.



The contents of the A-Register indicate the physical status of the equipment. (See STATUS REQUEST.)

The contents of the B-Register indicate the cause of the reject (bits 14–1 are zeros):

d = 1	The device or driver subroutine is busy and therefore unavailable, or, for Kennedy 1406 Tape unit, a broken tape condition encountered.
c = 1	A Direct Memory Access channel is not available to operate the device.
d = c = 0	The function or subfunction selected is not legal for the device.

2.1.4 BUFFER STORAGE AREA

The buffer address is the location of the first word of data to be written on an output device or the first word of a block reserved for storage of data read from an input device. The length of the buffer area may be specified in terms of words or characters. If the length is given as words, the value in the buffer length field must be a positive integer; if given as characters, a negative integer.†

In addition to describing the buffer area in the calling sequence, the area must also be specifically defined in the assembly language program, usually with a BSS or COM pseudo instruction.

2.2 ERROR CONDITIONS DURING EXECUTION

Illegal conditions encountered during .IOC. request processing are termed irrecoverable and cause a halt. (The halt is at the absolute location assigned to the symbol IOERR during Prepare Control System processing.) Diagnostic information is displayed in the A- and B-Registers at the time of the halt.

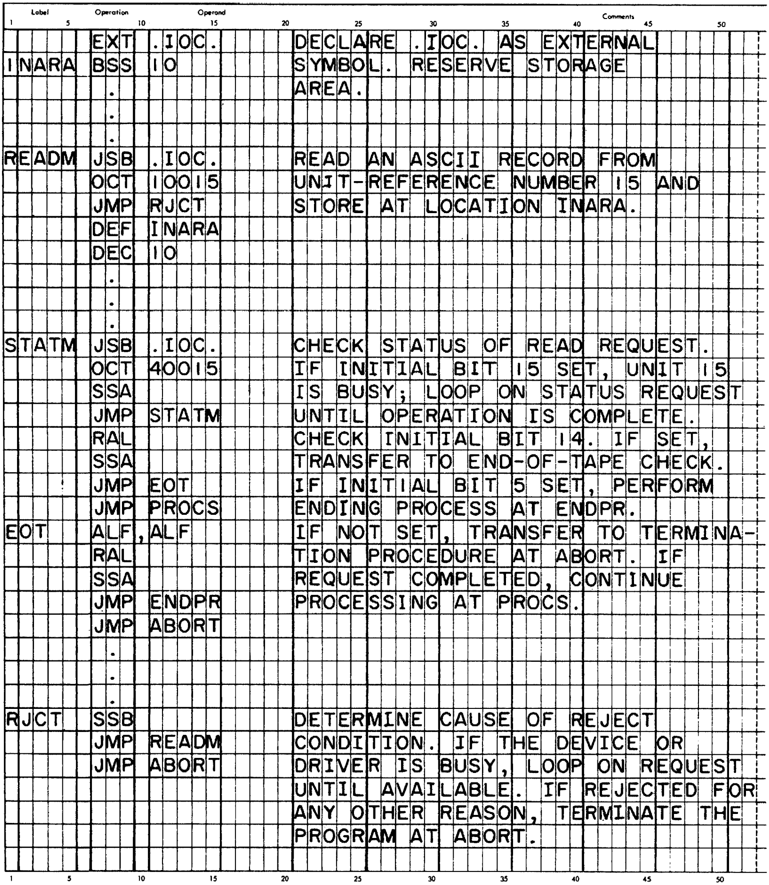
The B-Register contains the absolute location of the JSB instruction of the request call containing the illegal condition.

The A-Register contains a code defining the illegal condition:

A-Register	Explanation
000000	Illegal request code.

000001	Illegal unit-reference number in request.
000002	The Standard unit requested is not defined as a particular device in the Equipment Table.

Examples:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

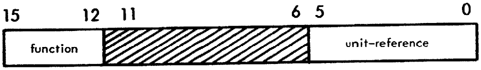
```
      EXT .IOC.     DECLARE .IOC. AS EXTERNAL
INARA BSS 10        SYMBOL. RESERVE STORAGE
                    AREA.
       .
       .
       .
READM JSB .IOC.     READ AN ASCII RECORD FROM
      OCT 10015     UNIT-REFERENCE NUMBER 15 AND
      JMP RJCT      STORE AT LOCATION INARA.
      DEF INARA
      DEC 10
       .
       .
       .
STATM JSB .IOC.     CHECK STATUS OF READ REQUEST.
      OCT 40015     IF INITIAL BIT 15 SET, UNIT 15
      SSA           IS BUSY; LOOP ON STATUS REQUEST
      JMP STATM     UNTIL OPERATION IS COMPLETE.
      RAL           CHECK INITIAL BIT 14. IF SET,
      SSA           TRANSFER TO END-OF-TAPE CHECK.
      JMP EOT       IF INITIAL BIT 5 SET, PERFORM
      JMP PROCS     ENDING PROCESS AT ENDPR.
EOT   ALF,ALF       IF NOT SET, TRANSFER TO TERMINA-
      RAL           TION PROCEDURE AT ABORT. IF
      SSA           REQUEST COMPLETED, CONTINUE
      JMP ENDPR     PROCESSING AT PROCS.
      JMP ABORT
       .
       .
       .
RJCT  SSB           DETERMINE CAUSE OF REJECT
      JMP READM     CONDITION. IF THE DEVICE OR
      JMP ABORT     DRIVER IS BUSY, LOOP ON REQUEST
                    UNTIL AVAILABLE. IF REJECTED FOR
                    ANY OTHER REASON, TERMINATE THE
                    PROGRAM AT ABORT.
```

2.3 CLEAR REQUEST

The CLEAR request can be used to terminate a previously issued input or output operation before all data is transmitted. It has the following form:

```
EXT         .IOC.
 .
 .
 .
JSB         .IOC.
OCT         <function> <unit-reference>
<normal return>
```

The second word consists of the following:



The function has the following value:

Function Name	Code (octal)
Clear	00

The only other parameter required is the unit-reference number. If the unit-reference number is specified as 00 (i.e., the second word of the calling sequence is OCT 0), all previous input and output operations are terminated. This request, the system CLEAR request, makes all devices available for the initiation of a new operation. On return from a system CLEAR request, the contents of the A- and B-Registers are meaningless.

Example:

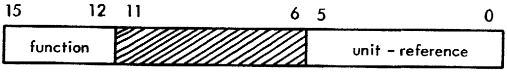
```
Label     Operation    Operand                Comments
1    5         10     15      20       25        30        35        40        45        50
       .
       .
       .
READM  JSB    .IOC.           READ AND PRINT A MESSAGE OF ONE
       OCT    10401           LINE FROM THE TELEPRINTER. WHEN
       JMP    REJ             CONTROL RETURNS AFTER INITIATING
       DEF    MSG             THE REQUEST, THE JSB MIGHT
       DEC    36              TRANSFER TO A SUBROUTINE WHICH
       JSB    TIMER           COULD CHECK THE TIME ALLOWED
       .                      FOR A MESSAGE TO BE COMPLETED.
       .
       .
CLRRD  JSB    .IOC.           IF THE MESSAGE IS NOT FURNISHED
       OCT    1               WITHIN A SPECIFIC TIME LIMIT, THE
       .                      REQUEST IS CLEARED BY THE SECOND
       .                      REQUEST TO .IOC.
       .
```

2.4 STATUS REQUEST

A request may be directed to .IOC. to determine the status of a previous input/output request or to determine the physical status of one or all units of equipment. The request has the following form:

```
JSB        .IOC.
OCT        <function><unit-reference>
<normal return>
```

The second word of the request has the following form:

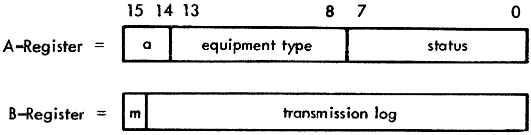


The function has the following value:

Function Name	Code (octal)
Status	04

The calling sequence requires no other parameters. A reject location is not necessary since the status information is always available. If the unit-reference number is specified as 00 (i.e., the second word on the calling sequence is OCT 40000), the request is interpreted as a system request.

If information is requested for a single unit, the Basic Control System returns to the location immediately following the request with the status information in the A and B registers:



a — Availability of device:

0 — The device is available; the previous operation is complete.

1 — The device is available; the previous operation is complete but a transmission error has been detected.

2 — The device is not available for another request; the operation is in progress.

equipment type — This field contains a 6-bit code that identifies the device referenced:

00-07 – Paper Tape devices
00 2752A Teleprinter
01 2737A Punched Tape Reader
02 2753A Tape Punch

10-17 – Unit Record devices
15 Mark Sense Reader

20-37 – Magnetic Tape and Mass Storage devices
- 20 Kennedy 1406 Incremental Tape Transport
- 21 HP 2020A Magnetic Tape Unit
- 22 HP 3030A Magnetic Tape Unit

40-77 – Instrumentation devices
- 40 Data Source Interface
- 41 DVM Programmer
- 42 Scanner Programmer
- 43 Time Base Generator

status

The status field indicates the actual status of the device when the data transmission is complete. The contents depend on the type of device referenced:

Device	Bits 7-0	Condition
Teleprinter reader or Punched Tape Reader	xx1xxxxx	End-of-Tape (10 Feed Frames)
Tape Punch	xx1xxxxx	Tape supply low
Kennedy 1406 Incremental Tape Transport	xx1xxxxx	End-of-Tape mark sensed
	xxxx1xxx	Broken tape; no tape on write head
	xxxxxxx1	Device busy
HP 2020 and 3030 Magnetic Tape Units =	1xxxxxxx	End-of-file record (17_8 for 2020, 23_8 for 3030) is detected or written.
	x1xxxxxx	Start-of-tape marker sensed
	xx1xxxxx	End-of-tape marker sensed
	xxx1xxxx	Timing error on read/write

xxxx1xxx I/O request rejected:

a. tape motion required but controller busy
b. backward tape motion required but tape at load point
c. write request given but reel does not have write enable ring.

xxxxx1xx Reel does not have write enable ring or tape unit is rewinding.

xxxxxx1x Parity error on read/write

xxxxxxx1 Unit busy or in LOCAL mode.

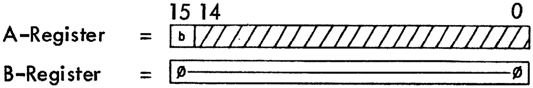
m = This bit defines the mode of the data transmission:

0 ASCII or BCD

1 Binary

transmission log = This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in calling sequence. The value is stored in this field only when the request is completed, therefore, when all data is transmitted or when a transmission error is detected.

If a system status request is made, the information in the A and B registers is as follows:



b = System Status
0 No device is busy
1 At least one device is busy

2.5 PAPER TAPE SYSTEM

2.5.1 RECORD FORMATS

The Paper Tape System operates on ASCII and binary records.

ASCII Records

An ASCII record is a group of characters terminated by an end-of-record mark which consists of a carriage return, CR , and a line feed, LF . If an odd number of characters is input, the last word transmitted to the buffer is padded with an ASCII blank.

For an input operation, the length of the record transmitted to the buffer is the number of characters or words designated in the request, or less if an end-of-record mark is encountered before the character or word count is exhausted. The codes for CR and LF are not transmitted to the buffer. An end-of-record mark preceding the first data character is ignored.

For an output operation, the length of the record is determined by the number of characters or words designated in the request. An end-of-record mark is supplied at the end of each output record by the input/output system.

If the last character of an ASCII output record to the teleprinter or punch is ←, however, the end-of-record mark is omitted. This allows control of teleprinter line spacing. For example, the user may write a message (the ← is not printed) and expect the reply to be typed on the same line. The reply must be terminated with the CR and LF .

If a RUB OUT code followed by a CR , LF is encountered on input from the teleprinter or Paper Tape Reader, the current record is ignored (deleted) and the next record transmitted. (RUB OUT appears on the teleprinter keyboard and is synonymous with the ASCII symbol DEL .)

If less than ten feed frames (all zeros) are encountered before the first data character from a paper tape input device, they are ignored. Ten feed frames are interpreted as an end-of-tape condition

Binary Records

A binary record is transmitted exactly as it appears in memory or an 8-level paper tape. The record length is specified by the number of characters or words designated in the request. The first character of a binary record must be non-zero. On input operations, less than ten feed frames preceding the first data character are ignored. Ten feed frames are interpreted as an end-of-tape condition (see STATUS REQUEST). On output, the system writes four feed frames to serve as a physical record separator.

Binary input records may be variable in length. The first word of the record contains a number in bits 15-8 specifying the length of the record (including the first word). The entire record including the word count is transmitted to the buffer. If the actual length exceeds the size of the buffer, only the number of words equivalent to the buffer length is transmitted.

NOTE: Although binary transmission is normally stated in words as opposed to characters, if an odd number of characters is requested on input the last word transmitted to the buffer is padded with binary zeros.

2.5.2 CALLING SEQUENCE

```
 EXT        .IOC.
  .
  .
  .
 JSB        .IOC.
 OCT        <function> <subfunction> <unit-reference>
{JSB}
{JMP}      <reject address>  <error return>
 DEF       <buffer address>
{DEC}
{OCT}      <buffer length>
 <normal return>
```

Function and Subfunction Codes

Allowable combinations of function and subfunction codes are as follows:

Operation	Octal value of Bits 15-6
Read ASCII record	0100
Read ASCII record and print	0104
Read binary record	0101
Read variable length binary record	0103

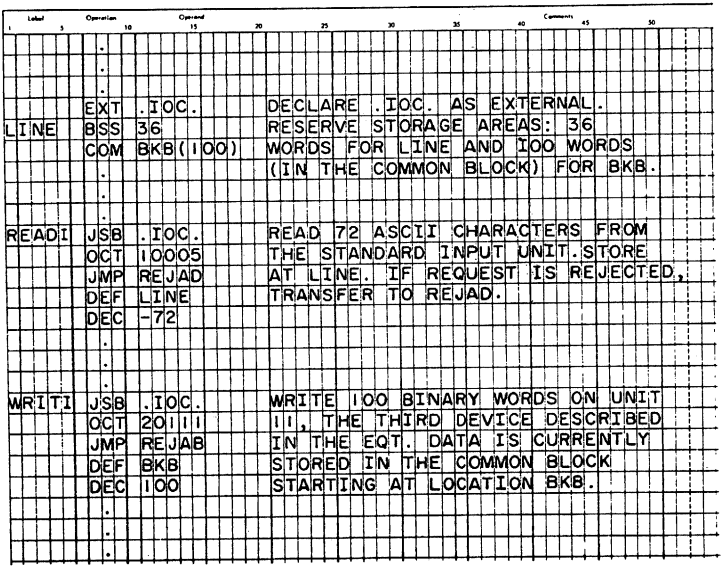
Write ASCII or BCD record	0200
Write binary record	0201

An illegal combination of codes is rejected.

Buffer Length

Character or word transmission may be specified for any paper tape device. The buffer length for data that may be printed on the teleprinter should be no more than 72 characters (36 words) or else the teleprinter will overprint at the end of line.

Examples:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
       .
       .
       .
      EXT  .IOC.      DECLARE .IOC. AS EXTERNAL.
LINE  BSS  36         RESERVE STORAGE AREAS: 36
      COM  BKB(100)   WORDS FOR LINE AND 100 WORDS
       .              (IN THE COMMON BLOCK) FOR BKB.
       .
       .
READI JSB  .IOC.      READ 72 ASCII CHARACTERS FROM
      OCT  10005      THE STANDARD INPUT UNIT. STORE
      JMP  REJAD      AT LINE. IF REQUEST IS REJECTED,
      DEF  LINE       TRANSFER TO REJAD.
      DEC  -72
       .
       .
       .
WRITI JSB  .IOC.      WRITE 100 BINARY WORDS ON UNIT
      OCT  20111      11, THE THIRD DEVICE DESCRIBED
      JMP  REJAB      IN THE EQT. DATA IS CURRENTLY
      DEF  BKB        STORED IN THE COMMON BLOCK
      DEC  100        STARTING AT LOCATION BKB.
       .
       .
       .
```

NOTE: In READI and WRITI, the leading 0 of the second word of the calling sequence need not be written in the source language since it is supplied in the object code as a result of using the OCT pseudo instruction.

2.6 HP 2891A CARD READER

2.6.1 DATA FORMATS

The HP 2891A Card Reader driver (D.11) provides three card reading functions to read any type of punched card, as described in the following paragraphs.

Hollerith to ASCII (Octal Equivalents) Conversion

Hollerith characters are converted to ASCII octal equivalents which are then placed into a buffer word according to the character's column number. All characters in odd-numbered columns are placed into the left byte (bits 15–8), and those in even-numbered columns are placed into the right byte (bits 7–0). The following table shows how the octal equivalent of each character appears in the two possible positions within a buffer word.

Hollerith to ASCII Octal Equivalents

Character			ASCII Octal Equivalent		Character			ASCII Octal Equivalent	
(name)	Hollerith	ASCII	Bits 15–8 (offset)	Bits 7–0 (true)	(name)	Hollerith	ASCII	Bits 15–8 (offset)	Bits 7–0 (true)
	A	A	404	101	(space)			200	040
	B	B	410	102		!	!	204	041
	C	C	414	103	(quote)	"	"	210	042
	D	D	420	104		#	#	214	043
	E	E	424	105		$	$	220	044
	F	F	430	106		%	%	224	045
	G	G	434	107		&	&	230	046
	H	H	440	110	(apostrophe)	'	'	234	047
	I	I	444	111		(	(	240	050
	J	J	450	112		)	)	244	051
	K	K	454	113		*	*	250	052
	L	L	460	114		+	+	254	053
	M	M	464	115	(comma)	,	,	260	054
	N	N	470	116	(hyphen or minus)	–	–	264	055
	O	O	474	117	(period)	.	.	270	056
	P	P	500	120		/	/	274	057
	Q	Q	504	121		:	:	350	072
	R	R	510	122		;	;	354	073
	S	S	514	123				360	074
	T	T	520	124		=	=	364	075
	U	U	524	125				370	076
	V	V	530	126		?	?	374	077
	W	W	534	127		@	@	400	100
	X	X	540	130	(cent)	¢		554	133
	Y	Y	544	131	(not mark)			564	135
	Z	Z	550	132	(vertical bar *)			570	136
	0	0	300	060	(underscore**)	_		574	137
	1	1	304	061		0-8-2		560	134
	2	2	310	062					
	3	3	314	063					
	4	4	320	064					
	5	5	324	065					
	6	6	330	066					
	7	7	334	067					
	8	8	340	070					
	9	9	344	071					

*numeric Y

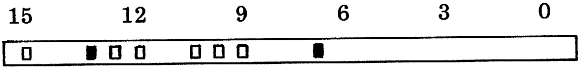
**numeric W

Consecutive characters (including blanks) are placed into consecutive buffer characters.

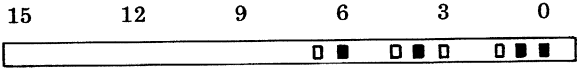
Read Hollerith to ASCII Function

The function code 0100 (READ HOLLERITH TO ASCII CONVERSION) reads a card containing "ASMB" in columns 1 through 4:

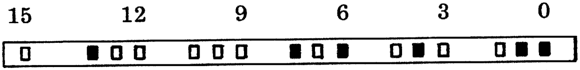
"A" = 101_8 which appears as 404_8 in "offset-octal" bits 15—8 of the first buffer word:



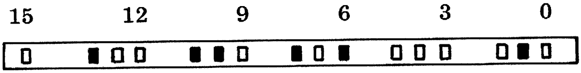
"S" = 123_8 in "true-octal" bits 7—0 of the first buffer word:



Thus the first packed word of the buffer is:



"M" = 115_8 which appears as 464_8 in "offset-octal" bits 15—8 of the second buffer word, and "B" = 102_8 in "true-octal" bits 7—0 of the second buffer word:

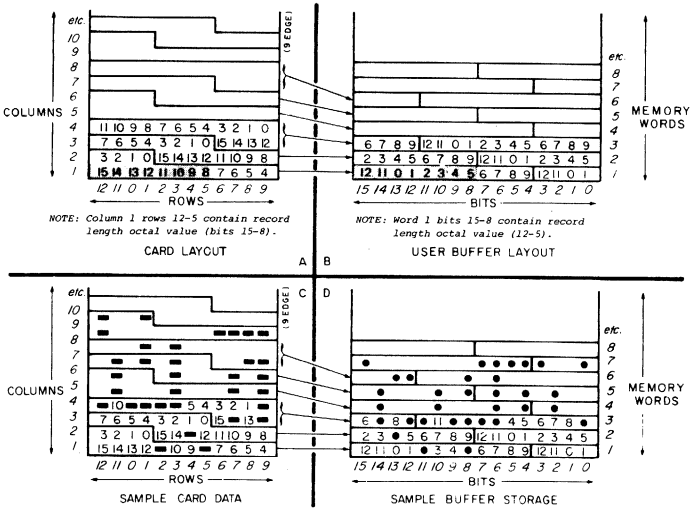


NOTE: Bits 8, 7, and 6 contain the octal sum of the least significant digit in the "offset-octal" value in bits 15—8 and the most significant digit in the "true-octal" value in bits 7—0.

Packed Binary

The Read Packed Binary function is used for cards punched in relocatable binary format by either an assembler or a compiler. The figure below shows how data is packed four card-columns into three buffer-words. One 80-column card fills 60 words of the user's buffer. Column 1 rows 12-5 in each card contain the Record Length octal value x, where $0 < x \leqslant 74_8$. See Appendix E Relocatable Tape Format.

Read Packed Binary Function



NOTE: Column 1 rows 12-5 contain record length octal value (bits 15-8).

CARD LAYOUT

NOTE: Word 1 bits 15-8 contain record length octal value (12-5).

USER BUFFER LAYOUT

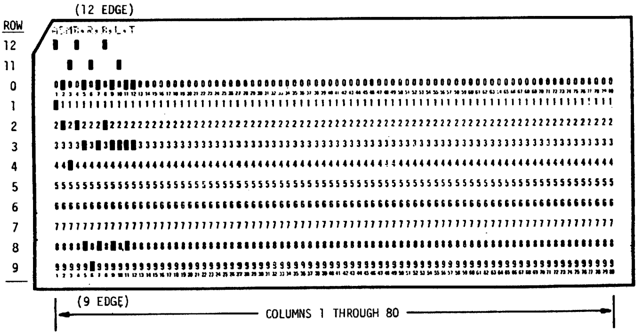
SAMPLE CARD DATA

SAMPLE BUFFER STORAGE

Column Image Binary

The Read Column Image Binary function places each 12-row card column into one 16-bit word of the user's buffer, right-justified. The four left bits (15–12) are set to 0, as shown below.

Read Column Image Binary Function



CARD DATA READ

ROW ASSIGNED:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	BITS
	(NONE)				12	11	0	1	2	3	4	5	6	7	8	9	ALL BUFFER WORDS
	(SET TO 0)				ONE COLUMN												

BUFFER WORD ASSIGNMENTS

CARD COLUMN																	BUFFER WORD
1	0	0	0	0	■	0	0	■	0	0	0	0	0	0	0	0	1st
2	0	0	0	0	0	0	■	0	■	0	0	0	0	0	0	0	2nd
3	0	0	0	0	0	■	0	0	0	0	■	0	0	0	0	0	3rd
4	0	0	0	0	■	0	0	0	■	0	0	0	0	0	0	0	4th
5	0	0	0	0	0	0	■	0	0	■	0	0	0	0	■	0	5th
6	0	0	0	0	0	■	0	0	0	0	0	0	0	0	0	■	6th
7	0	0	0	0	0	0	■	0	0	■	0	0	0	0	■	0	7th
8	0	0	0	0	■	0	0	0	■	0	0	0	0	0	0	0	8th
9	0	0	0	0	0	0	■	0	0	■	0	0	0	0	■	0	9th
10	0	0	0	0	0	■	0	0	0	■	0	0	0	0	0	0	10th
etc.																	etc.

BUFFER WORDS AFTER DATA TRANSFER

2.6.2 CALLING SEQUENCE

A calling sequence must be executed for each card read.

```
EXT      .IOC.
 .
 .
 .
JSB      .IOC.
OCT      <function><subfunction><unit reference number>
JMP      <reject address> }
DEF      <buffer address> }   [Omit for Clear or Status
                          }    requests.]
DEC }                     }
OCT }    <buffer length>  }

<normal return>
```

where:

<u>function</u> — in bits 15-12 } Function and Subfunction Codes
<u>subfunction</u> — in bits 11-6 } headings below.
<u>unit-reference number</u> — bits 5-0

reject address — .IOC. returns control to the user at this location if the function or subfunction request is rejected by the initiator section. The A-Register contains 1 and B-Register contains a cause-of-reject code:

a. If the card reader is busy or inoperable, or if the driver is busy, the B-Register contains 100000_8.

b. If the subfunction requested is invalid for the card reader, the B-Register contains 0; or if DMA is required but a DMA channel is currently not available, the B-Register contains 1.

buffer address — address of the first word of the user's buffer.

buffer length — a positive integer for 16-bit buffer words, or a negative integer for 8-bit buffer characters (half words). An odd number of characters specified is incremented by one (i.e., "–3" sets two buffer words, [3+1] /2 = 2). A 0 buffer length for either Read binary function feeds a card but ignores the data. A 0 buffer length for Read Hollerith to ASCII causes an immediate normal return with no action performed.

NOTE: Buffer characters (a negative integer) should be specified only with the READ HOLLERITH to ASCII function.

Function and Subfunction Codes

The allowable function and subfunction codes for the .IOC. calling sequence are shown below.

Function and Subfunction	Octal Code (bits 15–6)
Read Hollerith to ASCII octal equivalent conversion. Two characters per buffer word; see table on pages 2-16 and 2-17 and figure on page 2-18. Trailing blanks are suppressed.	0100
Read packed binary. Four 12-row card columns packed into three 16-bit buffer words; see figure on page 2-19. One 80-column card fills 60 sixteen-bit buffer words.	0103
Read column image binary. Each card column stored in one 16-bit buffer word right-justified; see figure on page 2-20. Bits 15-12 are set to 0.	0101
Clear request.	0000
Dynamic Status request. See Status.	0300
Status request. See Status.	0400

2.6.3 STATUS REQUESTS

Two types of status requests can be made: normal, which returns the status of the Card Reader for the last time it was referenced, and dynamic, which returns the actual status of the card reader.

A normal status request returns the current contents of EQT entry words 2 and 3 for the Card Reader in the A- and B-Registers, respectively. The table below shows the meanings of status bits 7-0 in the A-Register of EQT entry word 2. This driver returns an equipment type code of 11 in bits 13-8.

Status Bit Meanings

Bit (when set to 1)	Status Indicated
0	Reader not ready, or in TEST mode.
1	Illegal ASCII character(s), or hardware read trouble.
2	Card Reader in TEST mode.
3	Timing error, last column.
4	Pick failure.
5	Hopper empty.
6	Stacker is full.
7	End of file scratch is set and the feed hopper is empty.

When the user's calling sequence requests dynamic status, the driver returns only the status word in the A-Register (B-Register is unspecified). The status information is shown in the preceding table.

2.6.4 TRANSMISSION LOG

The transmission log in EQT entry word 3 for the Card Reader is a positive integer. It reports the number of buffer words or characters transmitted and the data transfer mode. (When bit 15 = 1, the mode is a Read Binary function; when bit 15 = 0, the mode is Read Hollerith to

ASCII.) According to the Read function requested, the transmission log count has one of three maximum values:

Function	Maximum Transmission Log Count
Read Hollerith to ASCII	80 characters or 40 words.
Read Packed Binary	60 words.
Read Column Image Binary	80 words.

The number of words or characters transmitted is determined by:

Function	Transmission Log Method
Read Hollerith to ASCII	The number of buffer characters requested, or the number of columns on the card, whichever is less, minus trailing blanks. (A totally blank card returns a zero.)
Read Packed Binary	The number of buffer words requested, or the octal number recorded in rows 12-5 of the first card column, whichever is less.
Read Column Image Binary	The number of words requested or the number of columns on the card, whichever is less.

2.7 HP 2778A, 2778A-001 LINE PRINTER

2.7.1 MODES OF OPERATION

This driver has three modes of operation: Plus, Normal, and TTY. The modes are selected by issuing the proper control subfunction or by selecting one of the following unit numbers at BCS configuration (PCS) time:

Unit No.	Mode
0 (Default Unit #)	Normal
2	Plus
4	TTY

For example, the Plus mode may be set at PCS time by supplying the following Equipment Table entry:

nn, D.12,U2

where nn is the ohannel number (select code) for the device.

In the Normal and Plus modes, the first character of the print buffer is used as control and is not printed. Instead, the second character of the buffer is printed in column one of the line printer paper.

In the Normal mode, if the first character is a "+", the driver interprets it as a blank (i.e., single space). In the Normal mode an attempt is made to drive the printer as a "space then print" device. Thus, if the command character says space 3 lines, the driver subtracts one and spaces 2 lines (one space was sent to terminate the last line, so the total is 3).

The Plus mode interprets a "+" in column one and overprints the current line on top of the last line. The driver sends a hold command at the end of each line and a single space before each line without a "+" in column one. The net effect is that the printer runs as a "space then print" device at approximately half-speed.

The TTY mode makes the Line Printer act like a teleprinter and prints the first character (in column one) of the buffer. Line space control for the TTY mode may be executed by using the print or control subfunction field. The TTY mode, if set, overrides the Plus and Normal modes and drives the printer as a "print then space" device. Two methods of spacing are permitted by using the print subfunction field.

The driver, in all modes, handles a line ending with a left arrow (←) by printing the first character in the buffer of the next request where the left arrow would have appeared had it been printed.

2.7.2 CALLING SEQUENCE

The general form of the input/output request is:

```
EXT     .IOC.
 .
 .
 .
JSB     .IOC.
OCT     <function><subfunction><unit-reference>
JMP     <reject address><error return>
DEF     <buffer address>

DEC }
OCT }   <buffer length>

<normal return>
```

2.7.3 INPUT/OUTPUT CONTROL (.IOC.)

All line printer input/output operations are requested by performing a JSB to entry point .IOC. The input/output control subroutine returns control to the calling program at the first location following the last word of the I/O request.

2.7.4 FUNCTION AND SUBFUNCTION CODES

The second word of the I/O request determines the function to be performed and the line printer unit-reference for which the action is to be taken. The bit combinations that comprise the control word are as follows:

15 12	11 6	5 0
function	subfunction	unit-reference

The function (bits 15-12) is the basic input/output operation; it may be any of the following:

Function Name	Code (Octal)
Clear	00
Write	02
Control	03
Status	04

Write Function (02)

Subfunction Bits (Ignore the x's)		Subfunction Description
00x	xxx	Normal and Plus mode — first character is carriage control, the ASCII character in the Allowable Motion Request table on page 2-30. The second character is printed in column one of the line printer.
01x	xxx	TTY mode — first character is data. The carriage control character is the low 6 bits of the status word (second word of equipment table). The status word is set with an extended carriage control explained below.
11x	ddd	TTY mode — first character is data. Carriage control is tape level corresponding to ddd in the Allowable Motion Requests table on page 2-30.
10x	xx0	Extended carriage control — first word in the buffer is sent as a carriage control command to the line printer. The first word is an octal code in bits 5–0, as defined in the Extended Carriage Control Code table on page 2-33. The buffer length (I/O request fifth word) should be set to 1.
10x	xx1	Extended carriage control — first word of buffer is set into status word to be used as TTY carriage control. The first word is an octal code in bits 5–0, as defined in the Extended Carriage Control Code table on page 2-33.

Control Function (03)

Subfunction Bits (Ignore the x's)		Subfunction Description
00x	000	Dynamic Status Request
00x	111	Clear TTY mode and Plus mode (and set Normal mode)
00x	110	Set TTY mode
00x	010	Set Plus mode
CC0	CCC	If not one of the above codes, CC1CCC will be sent to the line printer. (See the Allowable Motion Requests table on page 2-33.)

2.7.5 REJECT ADDRESS

Control is transferred to the third word of the I/O request if the input/output operation cannot be initiated. On transfer, the system provides status information which may be checked by the user's program. The A–Register is set to 0 to indicate that the operation is initiated, or is set to 1 to indicate that the operation is rejected. The B–Register contains the cause-of-reject code:

a. If the printer is busy or inoperable, or if the driver is busy, the B–Register contains 100000_8.

b. If the subfunction requested for the printer is invalid, the B–Register contains 0; or if DMA is required but a DMA channel is currently not available, the B–Register contains 1.

2.7.6 BUFFER STORAGE AREA

The buffer address is the location of the first word of data to be printed. The length of the buffer area may be specified in terms of words or characters. If the length is given as words, the value in the buffer length field must be a positive integer; if given as characters, a negative integer. A length of zero causes a blank line to be printed.

2.7.7 STATUS REQUESTS

Either of the following types of status requests may be made:

a. Normal status —

```
JSB      .IOC.
OCT      0400 <unit-reference>
<normal return>
```

b. Dynamic status —

```
JSB      .IOC.
OCT      0300 <unit-reference>
<normal return>
```

The dynamic status request is used to obtain the actual status of a line printer unit. The normal status request returns the status of the line printer unit for the last time it was referenced. The dynamic status request goes to the driver for its operation; it returns only the status word in the A-Register with nothing in particular in the B-Register. The EQT status table entry is updated by this request.

Allowable Motion Requests (HP 2778A, 2778A-001)

	Print Subfunction ddd code (octal)	Control Subfunction* CC0 CCC code (octal)	ASCII Character in Column One	Action
			0	Double space †
Printer Carriage Controls	7	67	1	Top of form †
	6	66	2	Bottom of form †
	5	65	3	Next sixth page †
	4	64	4	Next quarter page †
	3	63	5	Next half page †
	2	62	6	Next triple space line †
	1	61	7	Next double space line †
	0	60	8	Next single space line †
			9	Advance 55 lines
			:	Advance 54 lines
			;	Advance 53 lines
			<	Advance 52 lines
			=	Advance 51 lines
			>	Advance 50 lines
			?	Advance 49 lines
			@	Advance 48 lines
		47	A	Advance 47 lines
		46	B	Advance 46 lines
		45	C	Advance 45 lines
		44	D	Advance 44 lines
		43	E	Advance 43 lines
		42	F	Advance 42 lines
		41	G	Advance 41 lines
		40	H	Advance 40 lines
			I	Advance 39 lines
			J	Advance 38 lines
			K	Advance 37 lines
			L	Advance 36 lines
			M	Advance 35 lines
			N	Advance 34 lines
			O	Advance 33 lines
			P	Advance 32 lines

Print Subfunction ddd code (octal)	Control Subfunction* CC0 CCC code (octal)	ASCII Character in Column One	Action
	27	Q	Advance 31 lines
	26	R	Advance 30 lines
	25	S	Advance 29 lines
	24	T	Advance 28 lines
	23	U	Advance 27 lines
	22	V	Advance 26 lines
	21	W	Advance 25 lines
	20	X	Advance 24 lines
		Y	Advance 23 lines
		Z	Advance 22 lines
		[	Advance 21 lines
		\	Advance 20 lines
		]	Advance 19 lines
		↑	Advance 18 lines
		←	Advance 17 lines
		(Blank)	Advance 1 line
		!	Advance 15 lines
		”	Advance 14 lines
	04	#	Advance 13 lines
	03	$	Advance 12 lines
	02	%	Advance 11 lines
	01	&	Advance 10 lines
		’ (apostrophe)	Advance 9 lines
		(	Advance 8 lines
		)	Advance 7 lines
		*	Overprint next 1 line
		+	In Plus mode: overprint this line
		+	In Normal mode: Advance 1 line
		, (comma)	Advance 4 lines
		–	Advance 3 lines
		. (period)	Advance 2 lines
		?	Advance 1 line

* The x (priority bit 9) has been set = 0 for this table.

† These control requests include an automatic page eject.

Extended Carriage Control Code (HP 2778A, 2778A-001)

Octal Code (in bits 5–0)	Action
77	Top of Form †
76	Bottom of Form †
75	Next sixth page †
74	Next quarter page †
73	Next half page †
72	Next triple space line †
71	Next double space line †
70	Next single space line †
67	Advance 55 lines
66	Advance 54 lines
65	Advance 53 lines
64	Advance 52 lines
63	Advance 51 lines
62	Advance 50 lines
61	Advance 49 lines
60	Advance 48 lines
57	Advance 47 lines
56	Advance 46 lines
55	Advance 45 lines
54	Advance 44 lines
53	Advance 43 lines
52	Advance 42 lines
51	Advance 41 lines
50	Advance 40 lines
47	Advance 39 lines
46	Advance 38 lines
45	Advance 37 lines
44	Advance 36 lines
43	Advance 35 lines
42	Advance 34 lines
41	Advance 33 lines
40	Advance 32 lines
37	Advance 31 lines
36	Advance 30 lines
35	Advance 29 lines
34	Advance 28 lines
33	Advance 27 lines
32	Advance 26 lines
31	Advance 25 lines
30	Advance 24 lines
27	Advance 23 lines
26	Advance 22 lines
25	Advance 21 lines
24	Advance 20 lines
23	Advance 19 lines
22	Advance 18 lines
21	Advance 17 lines
20	Advance 16 lines
17	Advance 15 lines
16	Advance 14 lines
15	Advance 13 lines
14	Advance 12 lines
13	Advance 11 lines
12	Advance 10 lines
11	Advance 9 lines
10	Advance 8 lines
7	Advance 7 lines
6	Advance 6 lines
5	Advance 5 lines
4	Advance 4 lines
3	Advance 3 lines
2	Advance 2 lines
1	Advance 1 lines
0	Advance 0 line

† These actions include an automatic page eject.

Status Return Information

A-Register:

15 14	13 8	7 0
a	Equipment Type	Status

B-Register:

15	14 0
M	Transmission Log

a = Availability (A-Register bits 15 and 14):

0 = The device is available; the previous operation is complete.

1 = The driver is available; the operation could not be initiated because the device is not ready.

2 = The device is not available for another request; an operation is in progress.

Equipment Type (A-Register bits 13-8):

12_8 = HP 2778A (or 2778A-001) Line Printer

Status (A-Register bits 7-0):

Bits	Meaning
5-0	TTY termination code with bits 3-5 inverted
6	Left arrow (←) last time flag; if true, bit 6 = 1
7	Asterisk (*) last time flag; if true, bit 7 = 1

M = data transmission mode (B-Register bit 15):

Always 0 = ASCII

Transmission Log (B-Register bits 14—0):

This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in the I/O request.

2.7.8 CLEAR REQUEST

The clear request terminates a previously issued input or output operation and sets all busy flags to "not-busy." A clear request has the following form:

```
EXT     .IOC.
 .
 .
 .
JSB     .IOC.
OCT     0000 <unit-reference>
```

On return, the contents of the A- and B-Registers are meaningless. The clear request checks for multi-unit operation based on the device; i.e., the I/O channel number. The driver is cleared only if the clear request is for the current operation I/O channel.

If a clear request is issued while operating the driver in the plus mode, either of the following two events may occur:

1. If the driver is busy, the clear request will print and space one line.

2. If the driver is not busy, the clear request will not print and space one line.

In either case, the next print request following the clear request prints without spacing ("overprint next line" has been set by the driver); i.e., if the line printer paper is resting at Top-Of-Form and the driver is not busy, the first line of the next print request prints on the first line of the paper. However, if the line printer has just printed a line prior to the clear request and the driver is not busy, the first line of the next print request overprints the last line printed. To alleviate this problem, a control request may be issued prior to the print request.

2.7.9 ERROR CONDITIONS

Equipment Table Flags

Word 2 of the equipment table contains no hardware status in bits 7–0. See Status Return Information for the meaning of these bits.

Illegal Character

Should an illegal character be encountered, the driver will output an "@" character. A legal character is defined as $\geqslant 40_8$ and $\leqslant 137_8$ (all ASCII characters are legal), and all other octal numbers are considered to be illegal characters.

Illegal Buffer Length

Should an illegal buffer length be encountered, the driver will use 132 characters (or 66 words) as a legal length. A legal buffer length is defined as $\leqslant 132$ characters (or $\leqslant 66$ words).

2.8 HP 2767 LINE PRINTER

2.8.1 MODES OF OPERATION

The HP 2767 line printer driver has three modes of operation: Plus, Normal, and TTY. The modes are selected by issuing the proper control subfunction or by selecting one of the following unit numbers at BCS configuration (PCS) time:

Unit No.	Mode
0 (Default Unit #)	Normal
2	Plus
4	TTY

For example, the Plus mode may be set at PCS time by supplying the following Equipment Table entry:

nn, D.16,U2

where nn is the channel number (select code) for the device.

In the Normal and Plus modes, the first character of the print buffer is used as control and is not printed. Instead, the second character of the buffer is printed in column one of the line printer paper.

In the Normal mode, if the first character is a "+", the driver interprets it as a blank (i.e., single space). In the Normal mode an attempt is made to drive the printer as a "space then print" device. Thus, if the command character says space 3 lines, the driver subtracts one and spaces 2 lines (one space was sent to terminate the last line, so the total is 3).

The Plus mode interprets a "+" in column one and overprints the current line on top of the last line. The driver sends a hold command at the end of each line and a single space before each line without a "+" in column one. The net effect is that the printer runs as a "space then print" device.

The TTY mode makes the Line Printer act like a teleprinter and prints the first character (in column one) of the buffer. Line space control for the TTY mode may be executed by using the print or control subfunction field. The TTY mode, if set, overrides the Plus and Normal modes and drives the printer as a "print then space" device. Two methods of spacing are permitted by using the print subfunction field.

The driver, in all modes, handles a line ending with a left arrow (←) by printing the first character in the buffer of the next request where the left arrow would have appeared had it been printed.

2.8.2 CALLING SEQUENCE

The general form of the input/output request is:

```
EXT     .IOC.
 .
 .
 .
JSB     .IOC.
OCT     <function><subfunction><unit-reference>
JMP     <reject address><error return>
DEF     <buffer address>

DEC ⎫
OCT ⎭   <buffer length>

<normal return>
```

2.8.3 INPUT/OUTPUT CONTROL (.IOC.)

All input/output operations are requested by performing a JSB to entry point .IOC. The input/output control subroutine returns control to the calling program at the first location following the last word of the I/O request.

2.8.4 FUNCTION AND SUBFUNCTION CODES

The second word of the I/O request determines the function to be performed and the line printer unit-reference for which the action is to be taken. The bit combinations that comprise the control word as follows:

15 12	11 6	5 0
function	subfunction	unit-reference

The function (bits 15–12) is the basic input/output operation; it may be any of the following:

Function Name	Code (Octal)
Clear	00
Write	02
Control	03
Status	04

Write Function (02)

Subfunction Bits (Ignore the x's)		Subfunction Description
00x	xxx	Normal and Plus mode — first character is carriage control, the ASCII character in the Allowable Motion Requests table on page 2-41. The second character is printed in column one of the line printer.
01x	xxx	TTY mode — first character is data. The carriage control character is the low 6 bits of the status word (second word of equipment table). The status word is set with an extended carriage control explained below.
11x	ddd	TTY mode — first character is data. Carriage control is tape level corresponding to ddd in the Allowable Motion Requests table on page 2-41.
10x	xx0	Extended carriage control — first word in the buffer is sent as a carriage control command to the line printer. The first word is an octal code in bits 5–0, as defined in the Extended Carriage Control Code table on page 2-44. The buffer length (I/O request fifth word) should be set to 1.
10x	xx1	Extended carriage control — first word of buffer is set into status word to be used as TTY carriage control. The first word is an octal code in bits 5–0, as defined in the Extended Carriage Control Code table on page 2-44.

Control Function (03)

Subfunction Bits (Ignore the x's)		Subfunction Description
00x	000	Dynamic Status Request
00x	111	Clear TTY mode and Plus mode (and set Normal mode)
00x	110	Set TTY mode
00x	010	Set Plus mode
CCx	CCC	If not one of the above codes, CC1CCC will be sent to the line printer (See Allowable Motion Requests Table on page 2-41).

2.8.5 REJECT ADDRESS

Control is transferred to the third word of the I/O request if the input/output operation cannot be initiated. On transfer, the system provides status information which may be checked by the user's program. The A–Register is set to 0 to indicate that the operation is initiated, or is set to 1 to indicate that the operation is rejected. The B–Register contains the cause-of-reject code:

a. If the printer is busy or inoperable, or if the driver is busy, the B–Register contains 100000_8.

b. If the subfunction requested for the printer is invalid, the B–Register contains 0; or if DMA is required but a DMA channel is currently not available, the B–Register contains 1.

2.8.6 BUFFER STORAGE AREA

The buffer address is the location of the first word of data to be printed. The length of the buffer area may be specified in terms of words or characters. If the length is given as words, the value in the buffer length field must be a positive integer; if given as characters, a negative integer. A length of zero causes a blank line to be printed.

Allowable Motion Requests (HP 2767)

Print Subfunction ddd code (octal)	Control Subfunction** CCx CCC code (octal)	ASCII Character in Column One	Action
		0	Double space †
Printer Carriage Controls 7	67	1	Top of form †
6	66	2	Bottom of form †
5	65	3	Next sixth page †
4	64	4	Next quarter page †
3	63	5	Next half page †
2	62	6	Next triple space line †
1	61	7	Next double space line †
0	60	8	Next single space line †
		9	Advance 55 lines*
		:	Advance 54 lines*
		;	Advance 53 lines*
		<	Advance 52 lines*
		=	Advance 51 lines*
		>	Advance 50 lines*
		?	Advance 49 lines*
		@	Advance 48 lines*
	47	A	Advance 47 lines*
	46	B	Advance 46 lines*
	45	C	Advance 45 lines*
	44	D	Advance 44 lines*
	43	E	Advance 43 lines*
	42	F	Advance 42 lines*
	41	G	Advance 41 lines*
	40	H	Advance 40 lines*
		I	Advance 39 lines*
		J	Advance 38 lines*
		K	Advance 37 lines*
		L	Advance 36 lines*
		M	Advance 35 lines*
		N	Advance 34 lines*
		O	Advance 33 lines*
		P	Advance 32 lines*

Print Subfunction ddd code (octal)	Control Subfunction** CCx CCC Code (octal)	ASCII Character in Column One	Action
	27	Q	Advance 31 lines*
	26	R	Advance 30 lines*
	25	S	Advance 29 lines*
	24	T	Advance 28 lines*
	23	U	Advance 27 lines*
	22	V	Advance 26 lines*
	21	W	Advance 25 lines*
	20	X	Advance 24 lines*
		Y	Advance 23 lines*
		Z	Advance 22 lines*
		[	Advance 21 lines*
		\	Advance 20 lines*
		]	Advance 19 lines*
		↑	Advance 18 lines*
		←	Advance 17 lines*
		(Blank)	Advance 1 line *
		!	Advance 15 lines*
		"	Advance 14 lines*
		#	Advance 13 lines*
	04	$	Advance 12 lines*
	03	%	Advance 11 lines*
	02	&	Advance 10 lines*
	01	' (apostrophe)	Advance 9 lines*
		(	Advance 8 lines*
		)	Advance 7 lines*
		*	Overprint next line
		+	In Plus mode: overprint this line
		+	In Normal mode: Advance 1 line*
		, (comma)	Advance 4 lines*
		–	Advance 3 lines*
		. (period)	Advance 2 lines*
		/	Advance 1 line*

*Add six lines for any multiple skips crossing the paper perforations.
The HP 2767 pine printer will not print in three lines before and after the page perforations. Continuous listings are not possible with this printer.

Extended Carriage Contol Code (HP 2767)

Octal Code (in bits 5–0)	Action	Octal Code (in bits 5–0)	Action
77	Top of form †	37	Advance 31 lines
76	Bottom of form †	36	Advance 30 lines
75	Next sixth page †	35	Advance 29 lines
74	Next quarter page †	34	Advance 28 lines
73	Next half page †	33	Advance 27 lines
72	Next triple space line †	32	Advance 26 lines
71	Next double space line †	31	Advance 25 lines
70	Next single space line †	30	Advance 24 lines
67	Advance 55 lines	27	Advance 23 lines
66	Advance 54 lines	26	Advance 22 lines
65	Advance 53 lines	25	Advance 21 lines
64	Advance 52 lines	24	Advance 20 lines
63	Advance 51 lines	23	Advance 19 lines
62	Advance 50 lines	22	Advance 18 lines
61	Advance 49 lines	21	Advance 17 lines
60	Advance 48 lines	20	Advance 16 lines
57	Advance 47 lines	17	Advance 15 lines
56	Advance 46 lines	16	Advance 14 lines
55	Advance 45 lines	15	Advance 13 lines
54	Advance 44 lines	14	Advance 12 lines
53	Advance 43 lines	13	Advance 11 lines
52	Advance 42 lines	12	Advance 10 lines
51	Advance 41 lines	11	Advance 9 lines
50	Advance 40 lines	10	Advance 8 lines
47	Advance 39 lines	7	Advance 7 lines
46	Advance 38 lines	6	Advance 6 lines
45	Advance 37 lines	5	Advance 5 lines
44	Advance 36 lines	4	Advance 4 lines
43	Advance 35 lines	3	Advance 3 lines
42	Advance 34 lines	2	Advance 2 lines
41	Advance 33 lines	1	Advance 1 line
40	Advance 32 lines	0	Advance 0 line

†These actions include an automatic page eject.

2.8.7 STATUS REQUESTS

Either of the following types of status requests may be made:

a. Normal status —

```
JSB         .IOC.
OCT         0400 <unit-reference>
<return>
```

b. Dynamic status —

```
JSB         .IOC.
OCT         0300 <unit-reference>
<return>
```

The dynamic status request is used to obtain the actual status of a line printer unit. The normal status request returns the status of the line printer unit for the last time it was referenced. The dynamic status request goes to the driver for its operation; it returns only the status word in the A-Register with nothing in particular in the B-Register. The EQT status table entry is updated by this request.

Status Return Information

A-Register:

15 14	13 8	7 0
a	Equipment Type	Status

B-Register:

15	14 0
M	Transmission Log

a = Availability (A-Register bits 15 and 14):

0 = The device is available; the previous operation is complete.

1 = The driver is available; the operation could not be initiated because the device is not ready.

2 = The device is not available for another request; an operation is in progress.

Equipment Type (A-Register bits 13-8):

16_8 = HP 2767 Line Printer

Status (A–Register bits 7–0):

Bits	Meaning
5–0	TTY termination code with bits 3–5 inverted.
6	Left arrow (←) last time flag; if true, bit 6 = 1.
7	Asterisk (*) last time flag; if true, bit 7 = 1.

M = data transmission mode (B–Register bit 15):

Always 0 = ASCII

Transmission Log (B–Register bits 14–0):

This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in the I/O request.

2.8.8 CLEAR REQUEST

The clear request terminates a previously issued input or output operation and sets all busy flags to "not-busy". A clear request has the following form:

```
EXT     .IOC.

 .
 .
 .

JSB     .IOC.
OCT     0000 <unit-reference>
```

On return, the contents of the A- and B–Registers are meaningless. The clear request checks for multi-unit operation based on the device; i.e., the I/O channel number. The driver is cleared only if the clear request is for the current operation I/O channel.

If a clear request is issued while operating the driver in the plus mode, either of the following two events may occur:

1. If the driver is busy, the clear request will print and space one line.

2. If the driver is not busy, the clear request will not print and space one line.

In either case, the next print request following the clear request prints without spacing ("overprint next line" has been set by the driver); i.e., if the line printer paper is resting at Top-Of-Form and the driver is not busy, the first line of the next print request prints on the first line of the paper. However, if the line printer has just printed a line prior to the clear request and the driver is not busy, the first line of the next print request overprints the last line printed. To alleviate this problem, a control request may be issued prior to the print request.

2.8.9 ERROR CONDITIONS

Equipment Table Flags

Bits 14–9 of word one of the equipment table contain the line count of the HP 2767 Line Printer; i.e., if the carriage is resting on line 20, the bits contain 20B. Word 2 contains no hardware status in bits 7–0. See Status Return Information for the meaning of these bits.

Illegal Character

Should an illegal character be encountered, the driver will output an "@" character. A legal character is defined as $\geqslant 40_8$ and $\leqslant 137_8$ (all ASCII characters are legal), and all other octal numbers are considered to be illegal characters.

Illegal Buffer Length

Should an illegal buffer length be encountered, the driver will use 80 characters (or 40 words) as a legal length. A legal buffer length is defined as ≤80 characters (or ≤40 words).

2.9 KENNEDY INCREMENTAL TRANSPORT

2.9.1 RECORD FORMATS

Binary Coded Decimal Records

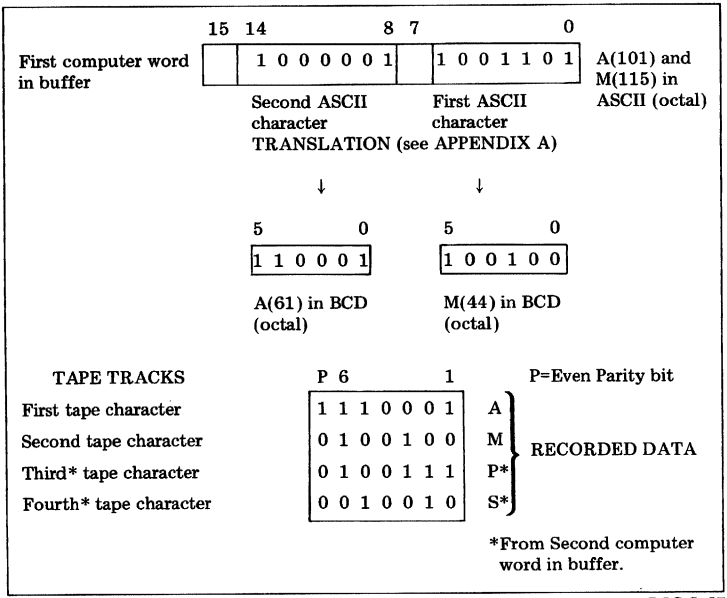
A BCD record is a group of BCD characters terminated (on magnetic tape) by a record gap. A request to write a BCD record results in the translation of each 7-level ASCII character in the buffer area into a 6-level BCD character on magnetic tape. (See Kennedy Incremental Transport BCD Record format heading below and the Kennedy ASCII-BCD Conversion on page A-2.) The translation process does not alter the original contents of the buffer.

The length of the record is determined by the number of characters or words designated in the request. A record gap is supplied at the end of each record by the input/output system.

If the last character in the buffer area is ←, however, the record gap is omitted. The ← is not written on tape.

A WRITE request specifying a buffer length of zero causes a record gap only to be written.

BCD Record Format - Kennedy Incremental Transport



2.9.2 CALLING SEQUENCE

```
EXT         .IOC.
  .
  .
  .
JSB         .IOC.
OCT         <function> <subfunction> <unit-reference>
JSB
JMP         <reject address> <error return>
DEF         <buffer address>
DEC
OCT         <buffer length>
<normal return>
```

2.9.3 FUNCTION AND SUBFUNCTION CODES

Allowable function codes for the 1406/1506 Kennedy Incremental Tape Transport are as follows:

WRITE (ASCII Mode only)	0200
WRITE End-of-file	0301
CLEAR	0000

2.10 MAGNETIC TAPE SYSTEM (HP 2020 MAGNETIC TAPE UNIT)

2.10.1 RECORD FORMATS

Binary Records

A binary record on magnetic tape is a group of 6-level tape "characters" recorded in odd parity and terminated by a record gap. The record length is determined by the number of characters or words in the buffer as designated in the request.

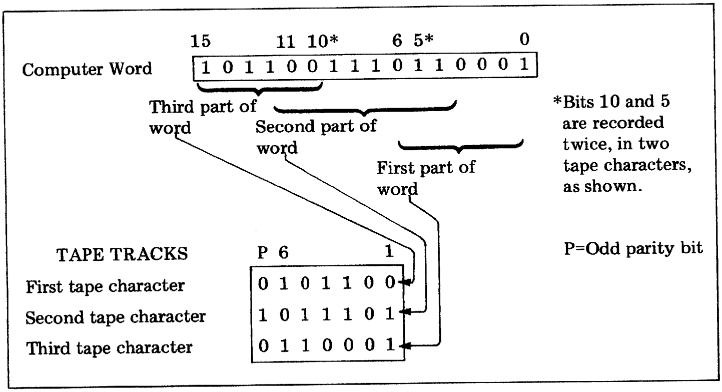
NOTE: Odd parity: a seventh bit is recorded on tape if the number of 1 bits in the six levels is an even decimal number (0, 2, 4 or 6).

Even parity: a seventh bit is recorded on tape if the number of 1 bits in the six levels is an odd decimal number (1, 3 or 5).

Each computer word is translated into three tape "characters" (and vice versa) as shown in the figure on page 2-51.

For output operations, the minimum buffer length is three computer words.

Binary Record Format (HP 2020)



*Bits 10 and 5 are recorded twice, in two tape characters, as shown.

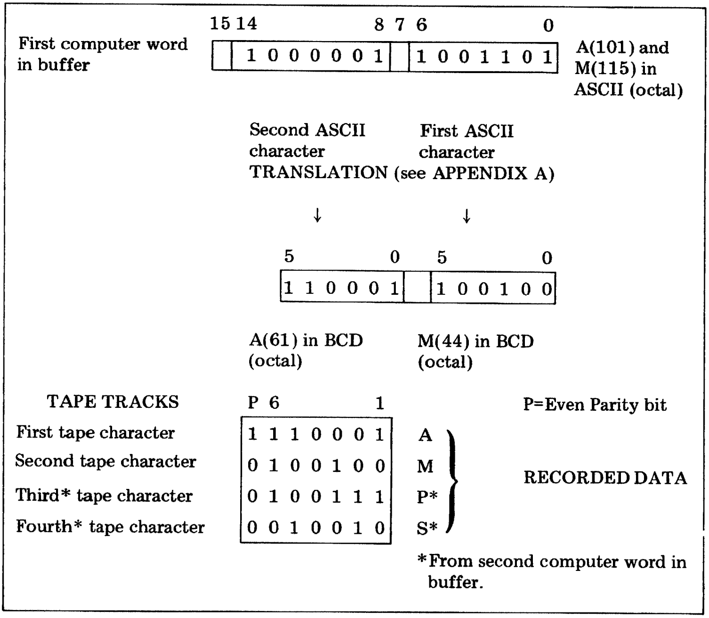
P=Odd parity bit

Binary Coded Decimal Records

A BCD record on magnetic tape is a group of BCD characters recorded in even parity and terminated by a record gap. A request to write a BCD record results in the translation of each 7-level ASCII character in the buffer area into a 6-level BCD character on magnetic tape. (Refer to the figure on page 2-52 and the table on page A-3.) A request to read a BCD record results in the translation of each BCD character into an ASCII character after the block has been read.

The length of the record may not be more than 120 characters. A record gap is supplied at the end of each record.

BCD Record Format (HP 2020)



2.10.2 CALLING SEQUENCE

```
EXT        .IOC.
  .
  .
  .
JSB        .IOC.
OCT        <function><subfunction><unit-reference>
JSB }
JMP }      <reject address><error return>
DEF        <buffer address>
DEC }
OCT }      <buffer length>
<normal return>
```

2.10.3 FUNCTION AND SUBFUNCTION CODES

All allowable combinations of function and subfunction codes are as follows:

Operation	Octal value of Bit 15-6
Read BCD record and convert to ASCII	0100
Read binary record	0101
Write BCD record after converting from ASCII	0200
Write binary record	0201
Write End-of-File (EOF) mark	0301
Forward space one record	0302
Backspace one record	0303
Rewind to start of tape (SOT) the LOAD Point, Ready (AUTO mode)	0304
Rewind to start of tape (SOT) the LOAD Point, Unload (LOCAL mode)	0305

2.10.4 BUFFER LENGTH

A WRITE request for the HP 2020 Magnetic Tape Unit must have a minimum buffer length of seven ASCII characters (four words). If less than seven characters are specified, spaces will be added to fill the seven characters.

2.11 MAGNETIC TAPE SYSTEM (HP 3030 MAGNETIC TAPE UNIT)

The 3030 Driver operates the HP 3030 9-channel magnetic tape controller. It initiates, continues and completes any tape operations requested through input/output control. As a module of the Basic Control System, the driver conforms to the general specifications for performing input/output under control of the Input/Output Control (IOC) module.

Two consecutive I/O channels are required with the data channel assigned to the higher priority of the two. The other channel is the command channel. Data is transferred to or from memory by a DMA channel.

The name of the Driver is D.22. The entry points are D.22 (Initiator Section) and C.22 (Continuator Section).

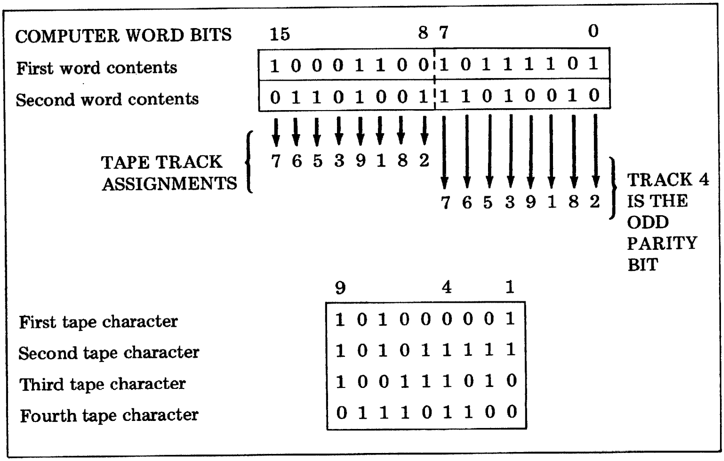
When configuring a BCS tape with the 3030 driver using PCS, the only requirement is a link from the command channel interrupt location to the entry point C.22 of the driver Interrupt Processor.

If an error is detected on a WRITE operation, the tape is backspaced over the record; three inches of tape are erased and the record is rewritten. This will continue until end-of-tape is sensed. If an error is detected on a READ operation, the driver will attempt to read ten times before aborting the operation.

2.11.1 RECORD FORMAT

Each computer word is translated into two tape "characters" by repositioning the bits as shown in figure below.

Record Format (HP 3030)



COMPUTER WORD BITS	15							8	7							0
First word contents	1	0	0	0	1	1	0	0	1	0	1	1	1	1	0	1
Second word contents	0	1	1	0	1	0	0	1	1	1	0	1	0	0	1	0
TAPE TRACK ASSIGNMENTS	7	6	5	3	9	1	8	2	7	6	5	3	9	1	8	2

TRACK 4 IS THE ODD PARITY BIT

	9					4			1
First tape character	1	0	1	0	0	0	0	0	1
Second tape character	1	0	1	0	1	1	1	1	1
Third tape character	1	0	0	1	1	1	0	1	0
Fourth tape character	0	1	1	1	0	1	1	0	0

2.11.2 CALLING SEQUENCE

```
EXT      .IOC.
 .
 .
 .
JSB      .IOC.
OCT      <function><subfunction><unit-reference>
JSB }
JMP }    <reject address><error return>
DEF      <buffer address>
DEC }
OCT }    <buffer length>
<normal return>
```

2.11.3 FUNCTION AND SUBFUNCTION CODES

All allowable combinations of function and subfunction codes are as follows:

Operation	Code
CLEAR	0000
READ (binary only)	0101 (or 0100)
WRITE (binary only)	0201 (or 0200)
DYNAMIC STATUS	0300
WRITE END-OF-FILE (EOF) MARK	0301
BACKSPACE ONE RECORD	0302
FORWARD SPACE ONE RECORD	0303
REWIND TO START OF TAPE (SOT, or the LOAD POINT), READY (AUTO mode)	0304
REWIND TO START OF TAPE (SOT, or the LOAD POINT), UNLOAD (LOCAL mode)	0305

2.11.4 BUFFER LENGTH

Character transmission is not applicable since the transmission is via a DMA channel. The minimum data block is twelve tape characters. Output blocks with a block length less than twelve characters are padded with zeros.

2.12 MAGNETIC TAPE SYSTEM (HP 7970 MAGNETIC TAPE UNIT)

2.12.1 CALLING SEQUENCE

```
EXT      .IOC.
 .
 .
 .
JSB      .IOC.
OCT      <function><subfunction><unit-reference>
JMP      <reject address>
DEF      <buffer address>
DEC      <buffer length>
<normal return>
```

where:

function (specified in bits 15–12)
subfunction (specified in bits 11–6)
Specifies the type of input/output operation being requested: Clear, Read, Write, Control, Status.

unit-reference (specified in bits 5–0)
Specifies the unit-reference number of the device used for input/output operations.

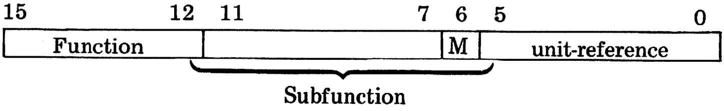
reject address
.IOC. returns control to the user at this location.

buffer address
Address of the first word of the user's buffer.

buffer length
The value in the buffer length field is specified in words (positive integer) or characters (negative integer). A buffer length of zero causes the driver to take no action on a write. A zero buffer length on binary read causes the driver to make a forward skip of one record, while a zero buffer length on ASCII read causes no action to be taken by the driver.

2.12.2 FUNCTION SUBFUNCTION, UNIT-REFERENCE CODES

The second word of the request determines the function to be performed and the MT unit-reference for which the action is to be taken.



If DMA is being used, the maximum I/O request must be no greater than the equivalent of 16,383 words.

NOTE: Setting the mode (m) bit 6 (on) causes the computer to transmit binary data as it appears in memory or on magnetic tape. Clearing the mode bit 6 (off) causes the computer to transmit ASCII data as it appears in memory or on tape.

2.12.3 REJECT ADDRESS

If the input/output operation cannot be performed, control is transferred to the third word of the calling sequence. When control is transferred, the computer system provides status information which can be checked by the user's program. The contents of the A-Register indicate the physical status of the equipment, and the contents of the B-Register indicate the cause of reject.

a. If bit 15 is 1, the driver is busy (unavailable).

b. If bit 0 is 1, a DMA channel is not yet available to operate the device.

c. If both bit 15 and bit 0 are 0, then the subfunction selected is illegal.

Allowable Motion Requests

Operation	Octal value of bits 15–6
Read ASCII record (RRF)	0100
Read Binary Record (RRF)	0101
Write ASCII record (WCC)	0200
Write Binary record (WCC)	0201
Write End-of-file mark (GFM)	0301
Backspace record (BSR)	0302
Forward space record (FSR)	0303
Rewind (REW)	0304
Rewind/Off Line (RWO)	0305
Erase four inches of tape (GAP)	0306
MTS Relocating Loader Skip record	0307
Forward Space Record (FSF)	0320
Backspace File (BSF)	0321
Status	0400
Clear	0000

Read and Write Requests

Bit 6 is only an indication of the request type; it does not imply two physical modes on the magnetic tape unit.

Rewind or Backspace Record Requests

This request performs no action if the tape unit addressed is at load-point. The status word indicates the SOT condition before and after the request is made.

Read Parity Error Conditions

The driver attempts to read a given record up to three times before declaring an irrecoverable parity error. If there is an irrecoverable parity error, the last try is transmitted to the user buffer and a normal completion return occurs. The status word indicates the parity and/or timing error.

Write Parity Error Conditions

The driver tries to rewrite a given record until either the record is successfully written or the end-of-tape is encountered.

Attempted Write Request

If a write request is made to a magnetic tape unit without a write enable ring, the driver makes an immediate completion return to the caller. Status bit 14 is set in the status word, causing the Formatter to print *EQR and halt. To proceed, insert a write enable ring in the magnetic tape.

Forward Motion Request

If forward motion is requested when the tape unit is at end-of-tape, the MT driver ignores the request and makes an immediate completion return. The exceptions to this situation are:

a. Write End-of-file mark request, and

b. Read record request.

Only one of these privileged requests can be made once the EOT has been encountered; after that, they are ignored by the driver.

Backward Motion Request (Rewind and Backspace Record and Backspace File)

This request restores the privileged nature of the write-end-of-file and read record requests.

Function/Subfunction Code Request 0307XX

Present in BCS MT drivers, the function/subfunction code request enables the Relocating Loader to operate within the Magnetic Tape System. If the request is followed by other I/O requests, they are treated as if the magnetic tape were not file-protected. The file protect feature is turned on again when the tape unit is rewound.

This request is identical to the forward space record request with the additional capability of spacing records within files 1 and 2 (even when the MT unit is in the protected file mode).

Backspace File Request and Forward Space File Request

These two requests cause the tape unit to go forward or backward until a file mark (EOF) is detected. Data is not transferred, and a parity error in any file record sets the parity error status bit.

The backspace file request positions the tape in front of a file mark or at load point, whichever comes first.

If the end-of-tape marker is sensed during execution of a forward space (record) request, the tape stops at the end of the current record. A status request should be used to check for this condition.

2.12.4 STATUS REQUESTS

As soon as tape movement operations for rewind and rewind/standby are initiated, the magnetic tape unit is available. The "A" field of a status reply is set to 00, enabling a system status request to indicate "not busy" for this EQT entry.

The normal status request returns the tape unit to the status when it was last referenced.

```
JSB        .IOC.
OCT        0400 <unit-reference>
<return>
```

The dynamic status request is used to obtain the actual status of a magnetic tape unit. It goes to the driver for operation and returns only the status word in the A–Register. The contents of the B–Register are not significant.

```
JSB        .IOC.
OCT        0300
<return>
```

Status Request Information

A–Register contents:

15 14	13 8	7 0
a	Equipment Type	Status

Bits 15–14 indicate the availability of the device (a):

If 0, the magnetic tape unit is available; previous operation is complete.

If 1, the magnetic tape unit is available; previous operation was ignored because either a write request was made without a write enable ring, or a tape motion request was made when the tape unit was off-line.

If 2, the magnetic tape unit is not available for another request; an operation is now in progress.

Bits 13–8 indicate the equipment type, i.e., specified as 23_8.

Bits 7–0 indicate the status of the device.

Bit	Condition
7	File Mark Sensed (EOF)
6	Load Point Status (BOT)
5	End-of-tape (EOT)
4	Data Timing Error
3	Command Rejected by the Controller
2	File Protected (no write enable ring)
1	Parity and/or Timing Error
0	Tape unit not on-line

NOTE: Bit 3 cannot be set using the driver.

B–Register contents:

15	14 0
m	Transmission Log

Bit 15, m, indicates the mode of data transmission (from the request)

If bit 15 = 0, ASCII code transmission
If bit 15 = 1, binary code transmission

Bit 14–0 indicate the transmission log, a field that is the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in the calling sequence of the read or write request. The driver cannot read or write an odd number of characters for this tape because the controller is a word device.

Minimum record length is one word.

An end-of-file mark record returns the user request length in the transmission log after being read, therefore allowing the binary read operation to operate properly through the Formatter. A write end-of-file mark returns one in the transmission log.

Control requests with a subfunction between 02 and 07 set the transmission log to zero.

Function requests of type 03 set m = 1.

2.12.5 CLEAR REQUEST

The clear request terminates a previously issued input or output operation before all data is transmitted. This request checks for multi-unit operation based on the device (i.e., I/O channel number). The driver is cleared only if the clear request is for the current operation I/O channel and physical unit number.

```
EXT      .IOC.
 .
 .
 .
JSB      .IOC.
OCT      0000 <unit-reference>
```

On return, the contents of the A- and B-Registers are not significant.

2.12.6 CONTROL REQUESTS

A request directed to .IOC. may also control the positioning of a reel on a magnetic tape device. The calling sequence is similar to the input/output request, but consists of only three words:

```
     EXT        .IOC.
      .
      .
      .
     JSB        .IOC.
     OCT        <function><subfunction><unit-reference>
     JSB }
     JMP }      <reject address><error return>
     <normal return>
```

The second word of the request has the following composition:

15 12	11 9	8 6	5 0
function	▨	subfunction	unit-reference

The function defines the calling sequence as a tape control request:

Function Name	Code (octal)
Position Tape	03

The subfunction defines the type of positioning:

Subfunction (octal)	Operation
0	dynamic tape status
1	write end-of-file
2	backspace one record
3	forward space one record
4	rewind
5	rewind and standby

As soon as tape movement operations (rewind, and standby) are initiated, the device is considered to be available; the "a" field of a status reply is set to 00 (see STATUS Request). The input/output driver is thus free to process requests for other devices. To obtain the actual status of the device when one of these commands has been issued, the dynamic tape status request is used. If the tape movement operation is still in progress the "a" field is set to 10.

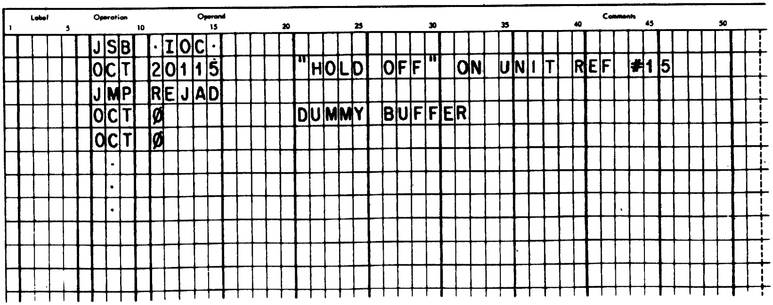
2.13 DATA SOURCE INTERFACE CALLING SEQUENCES

2.13.1 Binary Output Operation

A binary output operation causes the removal of "hold-off." The calling sequence is as below:

```
JSB          .IOC.
OCT          <function><subfunction><unit-reference>
JSB }
JMP }        <reject address><error return>
OCT          0     dummy buffer
OCT          0     buffer length
<normal return>
```

Example:



Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

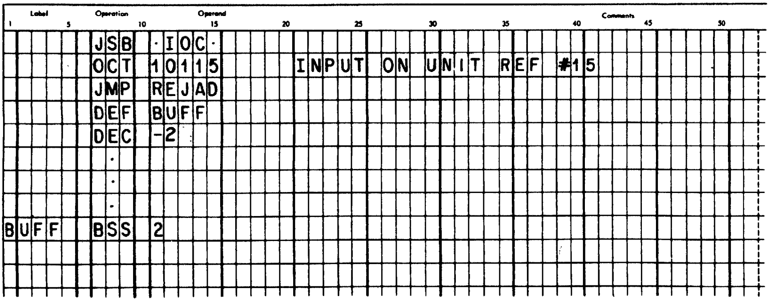
```
      JSB .IOC.
      OCT 20115     "HOLD OFF" ON UNIT REF #15
      JMP REJAD
      OCT Ø         DUMMY BUFFER
      OCT Ø
       .
       .
       .
```

2.13.2 Binary Input Operation

A binary input operation must have a 2-word buffer. Thirty-two bits (4 BCD characters) are read directly into the 2-word buffer.

```
                JSB   .IOC.
                OCT   <function><subfunction><unit-reference>
                JSB }
                JMP } <reject address><error return>
                DEF   <buffer address>
                DEC   4 (for 4 characters) or DEC-2 (for 2 words)
                 .
                 .
                 .
buffer address  BSS   2
```

Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      JSB .IOC.
      OCT 10115     INPUT ON UNIT REF #15
      JMP REJAD
      DEF BUFF
      DEC -2
       .
       .
       .
BUFF  BSS 2
```

2.13.3 ASCII Input Operation

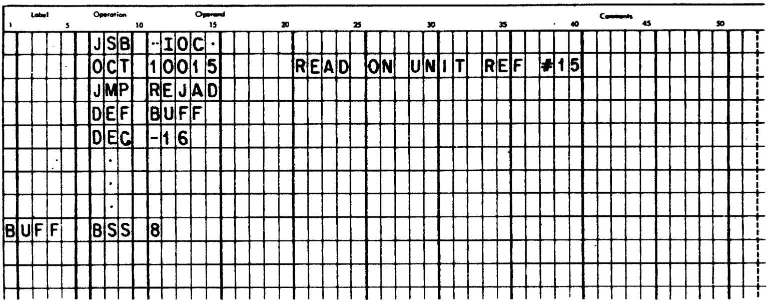
An ASCII input operation must have an 8 word buffer. Eight BCD characters are converted into 16 ASCII characters in the following format:

rf.$d_5d_4d_3d_2d_1d_{\emptyset}$E–ss∧∧gg

r	range — a negative power of 10
f	function
d_5–$d_{\emptyset}$	six digit data value
E–ss	range expressed as an exponent of two digits
∧∧	two blanks
gg	function expressed as a two-digit number

```
               JSB     .IOC.
               OCT     <function><subfunction><unit-reference>
               JMP  }
               JSB  }  <reject address>
               DEF     <buffer address>
               DEC     -16
                .
                .
                .
buffer address BSS     8
```

Example:



```
      JSB .IOC.
      OCT 10015     READ ON UNIT REF #15
      JMP REJAD
      DEF BUFF
      DEC -16
      .
      .
      .
BUFF  BSS 8
```

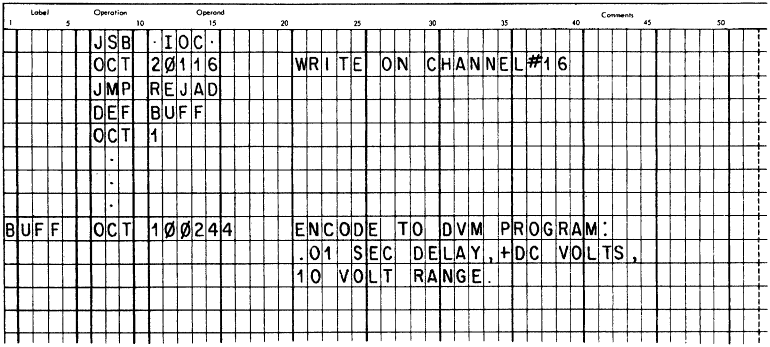
Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

2.14 DIGITAL VOLTMETER PROGRAMMER CALLING SEQUENCE

A WRITE request for the Digital Voltmeter Programmer requires that a one-word buffer be specified. This word contains the voltmeter program: sample period (bit 7-6), function (bits 5-3), and range (bits 2-0). If bit 15 contains a 1, an encode command is sent to the Voltmeter (bit 15 will always be 0 if the configuration includes a Scanner).

```
                JSB    .IOC.
                OCT    <function><subfunction><unit-reference>
                JSB  }
                JMP  } <reject address><error return>
                DEF    <buffer address>
                OCT    1
                <normal return>
buffer address  OCT    voltmeter program
```

Example:


Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

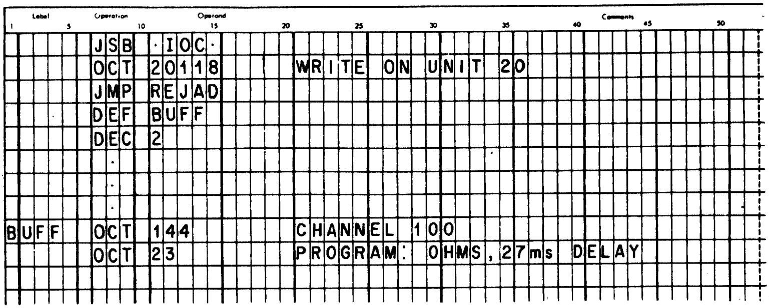
```
      JSB .IOC.
      OCT 2Ø116     WRITE ON CHANNEL#16
      JMP REJAD
      DEF BUFF
      OCT 1
       .
       .
       .
BUFF  OCT 1ØØ244    ENCODE TO DVM PROGRAM:
                    .01 SEC DELAY, +DC VOLTS,
                    10 VOLT RANGE.
```

2.15 SCANNER PROGRAMMER CALLING SEQUENCE

A WRITE request for the Scanner Programmer requires a 2-word buffer. The first word contains the channel number for the start of the scan. The second word contains the scanner program: the function (bits 4–3) and the delay (bits 2–0). The driver subroutine converts the binary channel number value produced by the Assembler to the BCD format required by the device.

```
                JSB   .IOC.
                OCT   <function><subfunction><unit-reference>
                JSB }
                JMP } <reject address><error return>
                DEF   <buffer address>
                DEC   2
                <normal return>
buffer address  OCT   xx   starting channel number
                OCT   xx   Scanner Program
```

Example:



Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      JSB .IOC.
      OCT 20118     WRITE ON UNIT 20
      JMP REJAD
      DEF BUFF
      DEC 2
       .
       .
       .
BUFF  OCT 144       CHANNEL 100
      OCT 23        PROGRAM: 0HMS, 27ms DELAY
```

2.16 INSTRUMENT CLEAR AND STATUS REQUESTS

2.16.1 INSTRUMENT CLEAR REQUEST

A CLEAR request on one of the instrument drivers follows the standard form:

```
JSB       .IOC.
OCT       <function><unit-reference>
<return>
```

where the function code = 00.

The request will result in the following conditions:

Data Source Interface	A CLEAR request causes no action. It is included for compatibility only.
Digital Voltmeter Programmer	A CLEAR request to this driver will remove the present program from the DVM but the program will not be destroyed.
Crossbar Scanner	A CLEAR request will inhibit the STEP or RESET command on the Scanner programmer driver.

2.16.2 INSTRUMENT STATUS REQUEST

No status information is available from the instrument drivers.

2.17 MARK SENSE CARD READER

The BCS Mark Sense Card Reader Driver D.15 operates the HP 2761A-007 Mark Sense Card Reader by initiating, continuing, and completing any operations through the Input/Output Control (.IOC.) subroutine within the Basic Control System (BCS). As a module of the BCS, this driver conforms to the general specifications for performing those controls.

The Initiator section of this driver interprets the function from the calling sequence, stores the buffer address and length, and signals the Mark Sense Card Reader to feed a card. An interrupt occurs for each clock mark printed on the 9-edge of the card to cause a JMP to the Continuator section of the driver. The Continuator then performs these tasks:

1. Saves the previous contents of all registers to be used by the Continuator section.
2. Examines bits 15–12 to check the validity of data. If the data is invalid, it determines why and sets status in the EQT table.
3. Ensures that only one card will be fed.
4. Stores the number of characters or words transmitted in the transmission log.
5. Restores the previously saved contents to the registers.
6. Terminates the transfer when the end-of-card is detected.

2.17.1 CALLING SEQUENCE

```
EXT       .IOC.
 .
 .
 .
JSB       .IOC.
OCT       <function><unit-reference>
JSB }
JMP }     <reject address><error return>
DEF       <buffer address>
DEC }
OCT }     <buffer length>
<normal return>
```

2.17.2 BUFFER LENGTH

The length can be specified for either words (a positive integer) or for characters (a negative integer) for any of the three reading functions. If either of the READ binary functions are requested and the buffer length specified is for an odd number of characters, the length will be effectively incremented by 1. Thus if 3 characters are specified, the buffer will be set for 2 computer words (i.e., (3+1)/2=2). If the buffer length is specified to be zero, a card is fed, but its data is ignored.

2.17.3 STATUS FIELD

The Status field indications are:

Bits 7-0	Condition
xxxxxxx1	Hopper empty or stacker full.
xxxxxx1x	Reader not READY.
xxxxx1xx	Pick failure.

The equipment type code is 15.

The transmission log has the following maximum values:

Function	Maximum Value
Read, Hollerith to ASCII	80 characters
Read, column image binary	80 words
Read, packed binary	60 words

2.17.4 FUNCTIONS

Function	Contents of bits 15–6
Read, Hollerith to ASCII (octal equivalent) conversion with two characters per computer word, as described in Appendix A. NOTE: In translating Hollerith to ASCII trailing zeros are suppressed.	0100
Read, packed binary; four 12-row card columns packed into three 16-bit computer words. Thus one 80-column card fills 60 words of the user's buffer. The packing format is described in the Small Programs Manual "BCS MARK SENSE DRIVERS, D.15" (HP 12602-90021).	0103
Read, column image binary; each card column is placed right justified into one 16-bit word. The four left bits (15–12) are set to zero, as shown in the Small Programs Manual.	
CLEAR request; allows the current card to finish feeding.	0000
STATUS request.	0400

RELOCATING LOADER 3

The Loader is the module of the Basic Control System that provides the capability of loading, linking, and initiating the execution of relocatable object programs produced by the Assembler, FORTRAN, and ALGOL. It is available in 4K and non-4K versions. ALGOL programs and the Relocatable Library stored on magnetic tape require the non-4K loader.

3.1 EXTERNAL FORM OF LOADER

The Loader, part of the tape titled "Configured BCS," is stored in an absolute record format on an external medium (on magnetic tape or 8-level paper tape) with the Input/Output Control subroutine (.IOC.) and the equipment driver subroutines. It is loaded by the Basic Binary Loader.

3.2 INTERNAL FORM OF LOADER

The Loader is located in high-numbered memory along with the Input/Output Control subroutine and the equipment driver subroutines. The Loader uses .IOC. for input/output operations; it refers to the Standard input and output units. The binary object program is read from the Standard Input unit; comments to the user (e.g., Loader diagnostics) are written on the Teleprinter Output unit; and library routines referenced by the object program are assumed to be on the Program Library unit.

3.3 RELOCATABLE PROGRAMS

The process of assembling or compiling a set of symbolic source program statements results in the generation of relocatable object code. Relocatable code assumes a starting location of 00000. Location 00000 is termed the relative, or relocatable origin. The absolute origin (also termed the relocation base) of a relocatable program is determined by the Loader. The value of the absolute origin is added to the zero-relative value of each operand address to obtain the absolute operand address. The absolute origin, and thus the values of every operand address, may vary each time the program is loaded.

A relocatable program may be made up of several independently assembled or compiled subprograms. Each of the subprograms would have a relative origin of 00000. Each subprogram is then assigned a unique absolute origin upon being loaded. Subprograms executed as a single program may be loaded in any order. The absolute origins will differ whenever the order of loading differs.

The operand values produced by the Assembler, FORTRAN, or ALGOL may be program relocatable, base page relocatable, or common relocatable. Each of these segments of the program has a separate relocation base or origin. Operands that are references to locations in the main portion of the program are incremented by the program relocation base; those referring to the base page, by the base page relocation base; and those referring to common storage, by the common relocation base.

If the Loader encounters an operand that is a reference to a location in a page other than the "current" page or "base" page, a link is established through the base page. A word in the base page is allocated to contain the full 15-bit address of the referenced location. The address of the word in the base page is then substituted as an indirect address in the instruction in the "current" page. If other similar references are made to the same location, they are linked through the same word in the base page.

3.4 RECORD TYPES

The Loader processes three to five record types for a program. These record types are produced by the Assembler, FORTRAN, or ALGOL in the following sequence:

NAM	Name record
ENT	Entry point record
EXT	External name record
DBL	Data block record
END	End record

The NAM, DBL, and END records exist for every object program; ENT and EXT appear only if the corresponding pseudo instructions are used in the source program.

NAM

The NAM record contains the name of the program and the length of the main, base page, and common segments. The NAM record signifies the beginning of the object program.

ENT

The ENT record defines the names of 1 to 14 entry points within this program. Each of the four-word entries in the record contains the name, the relocatable address of the name; and an indicator which specifies whether the address is program or base page relocatable.

EXT

The EXT record contains from 1 to 19 three-word entries which specify the external references defined in the program. The three words allow a maximum of five ASCII characters for the symbol and a number used by the Loader to identify the symbol.

DBL

A DBL record contains 1 to 45 words of the object program. It indicates the relative starting address for the string of words and whether this portion of the object code is part of the main program or base page segment. For each of the words there is also a relocation indicator which defines the relocation base to be applied to each operand value. Possible relocation factors are:

Absolute — Operand is an absolute expression or constant. There is no relocation base.

15-bit Program Relocatable — Operand is a 15-bit value to which is added the program relocation base.

15-bit Base Page Relocatable — Operand is a 15-bit value to which is added the base page relocation base.

15-bit Common Relocatable

Operand is a 15-bit value to which is added the common relocation base.

External Symbol Reference

Operand is a reference to an external symbol. Value is supplied when the Loader determines the absolute location of the linkage word in the Base Page which contains the 15-bit address of the related entry point.

Memory Reference Instruction

A memory reference instruction in the form of a two-word group which consists of the instruction code, a full 15-bit operand address, and a relocation indicator for the operand address. The relocation indicator can define the operand address to be program, base page, or common relocatable.

END

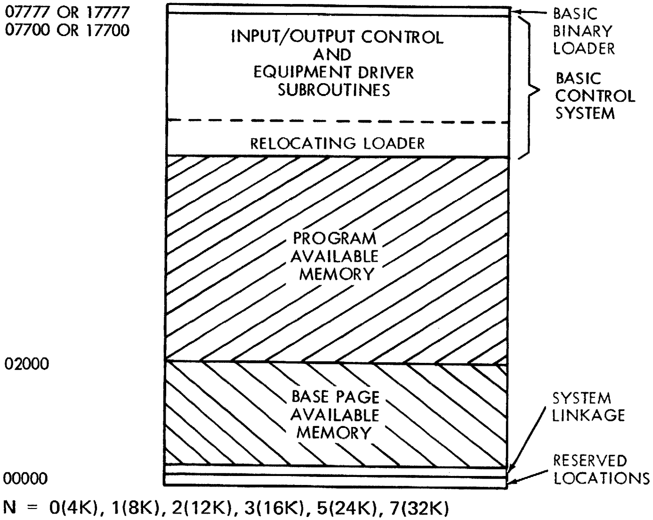
The END record terminates the block of records in an object program. The END record may contain a 15-bit address which is the location to which control is transferred by the Loader to begin program execution.

3.5 MEMORY ALLOCATION

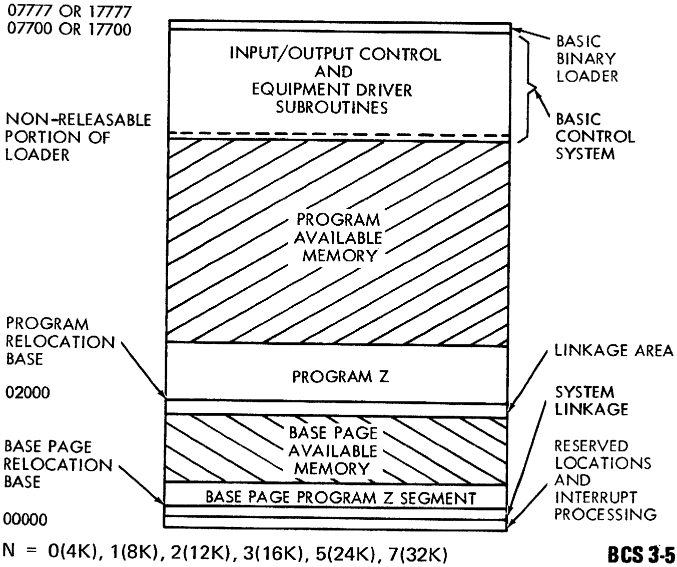
The Loader loads the object program into available memory. Available memory is defined as that area of memory not allocated for hardware and system usage. Available memory is divided into two segments:

> Available Memory in Base Page – used for the operand linkage area, program blocks origined into the Base Page by the Assembler pseudo instruction ORB, and for program blocks assigned to the Base Page by the Loader when the amount of program available memory is insufficient.
>
> Program Available Memory – used for the main body of the program and may be used by the common block should the area used by the Loader be insufficient.

Prior to loading the object program, memory is allocated as follows:



Assuming Program Z is to be loaded and executed – after loading, the memory might be allocated as follows:

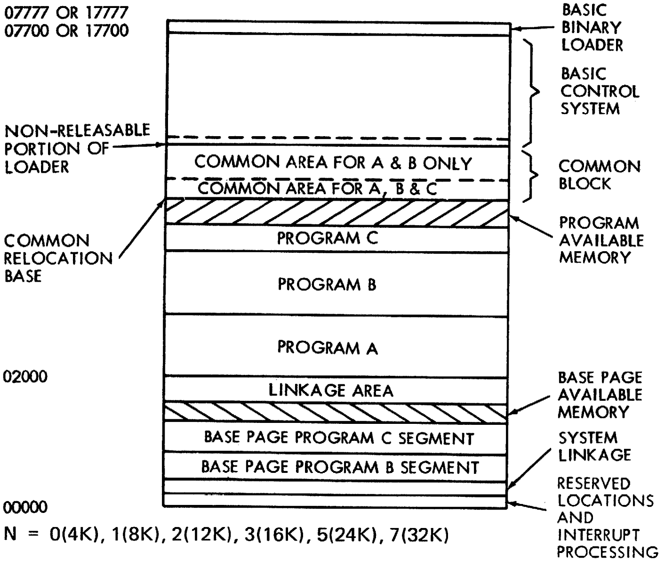


07777 OR 17777
07700 OR 17700
NON-RELEASABLE PORTION OF LOADER
INPUT/OUTPUT CONTROL AND EQUIPMENT DRIVER SUBROUTINES
PROGRAM AVAILABLE MEMORY
PROGRAM RELOCATION BASE
PROGRAM Z
02000
BASE PAGE RELOCATION BASE
BASE PAGE AVAILABLE MEMORY
BASE PAGE PROGRAM Z SEGMENT
00000
BASIC BINARY LOADER
BASIC CONTROL SYSTEM
LINKAGE AREA
SYSTEM LINKAGE
RESERVED LOCATIONS AND INTERRUPT PROCESSING
N = 0(4K), 1(8K), 2(12K), 3(16K), 5(24K), 7(32K)

Options selected during PCS processing can define the equipment driver subroutines and other routines as external routines which must be satisfied at run-time. If selected, these routines would be allocated to the available memory areas, and the length of the absolute segment of BCS reduced accordingly.

If several programs are to be loaded and executed together, the following might occur:

> Assume three programs, A, B, and C, comprise a running program. Programs A and B share a common block, a portion of which is also shared by C. Programs B and C contain segments which are designated to be allocated to the Base Page. Allocation is as follows:


07777 OR 17777
07700 OR 17700
NON-RELEASABLE PORTION OF LOADER
COMMON RELOCATION BASE
02000
00000
N = 0(4K), 1(8K), 2(12K), 3(16K), 5(24K), 7(32K)
COMMON AREA FOR A & B ONLY
COMMON AREA FOR A, B & C
PROGRAM C
PROGRAM B
PROGRAM A
LINKAGE AREA
BASE PAGE PROGRAM C SEGMENT
BASE PAGE PROGRAM B SEGMENT
BASIC BINARY LOADER
BASIC CONTROL SYSTEM
COMMON BLOCK
PROGRAM AVAILABLE MEMORY
BASE PAGE AVAILABLE MEMORY
SYSTEM LINKAGE
RESERVED LOCATIONS AND INTERRUPT PROCESSING

Common Block Allocation

The first common length declaration (i.e., the first program containing a common segment) processed by the Loader establishes the total common storage allocation in high memory overlaying the major portion of the area occupied by the Loader. Subsequent programs must contain common length declarations which are less than or equal to the length of the first declaration.

To allocate the common area, the Loader subtracts the total length of the block from the address of the last releasable word in the Loader. The resulting memory address +1 is the origin of the common block. This value is used throughout the entire loading process as the common relocation base.

Program Storage

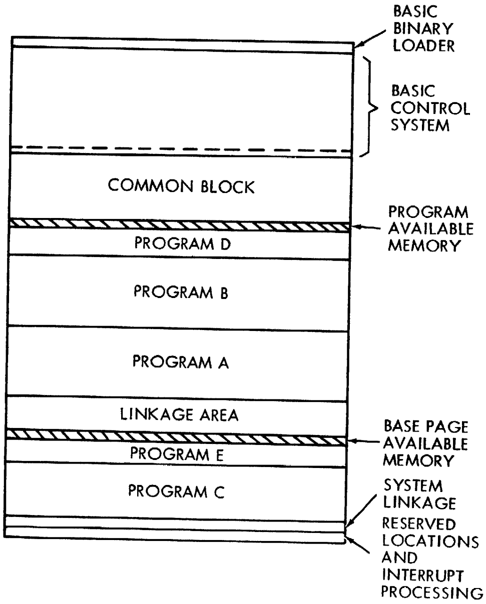
The program length is compared with the amount of available memory. If sufficient space is available, the program is loaded and the upper and lower bounds recorded. If the program has a base page segment, or if the program consists entirely of coding to be stored in the base page, the length of the segment is compared to the amount of available base page memory. If there is enough space in this area, the segment is loaded and the bounds recorded. Whatever is loaded first is usually originated at absolute location 02000 (page 1, module 0). The initial base page segment is usually originated immediately following the area set aside for reserved locations, interrupt processing, and system linkage. Subsequent main program and base page segments are loaded into the next available higher numbered areas contiguous with the previously loaded segments.

Providing the memory allocation list option is selected, the name of each program, its upper and lower bounds, and its base page upper and lower bounds are printed after the program is loaded. The format is as follows:

```
<program name>
lllll uuuuu (main program bounds)
lllll uuuuu (Base Page bounds)
```

If the Loader finds that the main program segment about to be loaded can not fit in the memory area available for the main segment, it compares the segment's length to the length of available memory in the base page. If there is sufficient space, the main segment will be loaded in the base page. The next segment will be loaded in the main program area if it will fit, or in the base page if not (providing there is sufficient space in the base page). When all available base page space has been used, loading is terminated.

For example, assume that several programs are to be loaded in sequence A, B, C, D, E, and have sizes such that they can not all fit in the main program available memory.



3.6 OBJECT PROGRAM RECORD PROCESSING

ENT/EXT Record Processing

The Loader constructs and maintains a Loader Symbol Table which contains entry points and external symbols which are declared in the programs and entry point names of any BCS system subroutines that have been defined as relocatable. As each entry point is encountered its relocated (absolute) address is recorded in the table. As each external reference is processed, a link word is established in the Base Page. The general processing of the entries in an ENT and EXT record involves searching the loader symbol table to locate a match between the symbols. When a match is found, the absolute entry point address is stored in the base page link word.

The Loader assumes that there is a user program, BCS system routine, or Relocatable Library routine entry point for every external reference. If none exists, the external reference is undefined and considered to be in error. A list of undefined external symbols is printed at the end of the loading operation. If duplicate entry points are detected, a diagnostic is issued. For duplicate entry points, only the first routine is accepted.

Each entry in the Loader Symbol Table occupies five words. The Table is positioned before the beginning of the Loader and extends backwards toward low-numbered memory. If sufficient space is not available in the main program portion of memory to store a five-word entry, a diagnostic message is issued and the loading operation is terminated.

DBL Record Processing

A loadaddress for the data or instruction words in a DBL record is relocated by adding either the program relocation base or the base page relocation base. The resulting value is the absolute address for storing the first word. The second word is stored at address +1, the third at address +2 and so forth. A relocation base is added to each operand address as specified by the relocation indicator.

The processing for an external reference word involves a search of the Loader Symbol Table for the related entry. When found, the address of the link location in the Base Page is extracted and stored as an indirect address in the instruction.

When a memory reference instruction is processed, the Loader first applies the proper relocation base, (program, base page, or common) to the 15-bit operand address. If the resulting absolute operand address references the Base Page, the address (bits 09-00) is set into the operand field and the instruction is stored in memory at the current load address. When the absolute operand address and the current load address are in the same page, the operand address is truncated to bits 09-00 and set as the instruction operand address. If the operand address is in a page other than the current load address page, the operand address is stored in the Linkage area of the Base Page and a reference to this location set as an indirect address in the operand field of the instruction.

A memory overflow condition can occur when insufficient space is available in the base page to allocate a linkage word. A diagnostic message is issued and the loading operation is terminated.

END Record Processing

When an END record is encountered, the Loader determines if it contains a transfer of control address. If it does, the address is saved.

If loading is from the Relocatable Library and no undefined external references exist, the end-of-loading operation is performed.

If loading is from the standard input unit or Relocatable Library unit and if undefined external references exist, the Loader requests the next record. If the next record is a NAM record, processing of the next program begins. If the result of the request is an end-of-information indication, an End condition exists.

Relocatable Library Loading

Loading from the Relocatable Library differs from loading user programs. Only those programs in the library that contain entry points matching undefined external symbols in the loader symbol table are loaded. After each library program is loaded, the loader symbol table is checked for undefined symbols. If none exist, the loading operation is complete and the program is ready to be executed.

End Condition

When the Loader requests input and end-of-tape occurs on the input device, an End condition exists. The Loader acknowledges this condition by writing the message "LOAD" on the teleprinter. The user responds to this message by setting switches 2–0 of the Switch Register. (See "Loader Operating Procedures.") Four replies are available:

a. Load next program from standard input unit. External BCS system subroutines are considered to be part of the program and must be loaded from the standard input unit (unless they are made part of Relocatable Library tape).

b. All programs are loaded; proceed to the end-of-loading operation.

c. Terminate loading operation. This forces program execution even through there may be undefined external references.

d. Load from Relocatable Library; all user programs are loaded.

End-of-Loading Operation

The end of loading is signaled by the second or fourth response to an End Condition. The Loader then searches the Loader Symbol Table for any undefined external references. Any such undefined external symbols are written on the Teleprinter Output unit and the "LOAD" message is repeated.

When the loading operation is completed or when the user has requested termination of the loading process, the Loader produces a memory allocation list. (This list may be omitted; see "Loader Operating Procedures.") The format of the list is as follows:

```
<symbol 1>   aaaaa

<symbol 2>   aaaaa

     .
     .
     .

<symbol n>   aaaaa
```

The symbols are the entry points in the user's program, the Basic Control System, or the Relocatable Library and the a's are their absolute addresses.

If a common block was allocated, the lower and upper bounds of the block are listed as follows:

```
*COM      lllll      uuuuu
```

The bounds of the Linkage Area are listed as follows:

```
*LINKS     lllll     uuuuu
```

The l's are the absolute lower bounds and the u's are the absolute upper bounds.

3.7 PROGRAMMING CONSIDERATIONS

When a program has been completely loaded, its execution is initiated by performing a Jump Subroutine to the transfer address (from the last END record containing an address). The initial contents of the transfer address should be a NOP, OCT 0, etc., not the first executable instruction of the program.

3.8 LOADER OPERATING PROCEDURES

The exact operating procedures for the loader depend on the available hardware configuration and the construction of the Basic Control System through use of the Prepare Control System routine. The user should know the assignment of input/output equipment and memory size before using the Loader.†

Loading Options

The Basic Control System Loader loads one or more tapes containing relocatable programs. The message "LOAD" is typed when an end-of-tape condition is encountered. The user then loads the next tape, indicates loading from the Relocatable Library, specifies that loading is complete, etc. When all programs are loaded and no undefined external references remain, the Loader types the message "*LST" allowing the user to bypass part of the memory allocation list. Following the response, the Loader types the message "*RUN." The user then initiates program execution.

Memory Allocation List

A memory allocation list can be obtained for the programs being loaded. The list includes the name, main program bounds, and base page bounds for each program. At the completion of the loading operation, this portion of the list may be followed by a list of all entry points and their absolute addresses, the bounds of the common block, and the bounds of the linkage area. The Switch 15 setting determines the contents of the list.

To obtain the bounds for each program on a tape, Switch 15 must be set to 0 before the tape is loaded (in response to the "LOAD" message). To bypass the program bounds listing, set Switch 15 to 1 before loading the tape. The switch setting may be altered whenever the "LOAD" message is typed.

To obtain the entry point list, the common bounds, and the linkage area bounds, set Switch 15 to 0 in response to the message "*LST", which is printed after all programs are loaded. To bypass this portion of the list, set Switch 15 to 1.

Absolute Binary Output

When it is necessary to utilize the area occupied by the Loader for program storage or when an absolute version is desired for "production stage" programs, the user may specify that an absolute binary tape be punched. The process involves a simulated loading operation; however, the absolute program is punched on tape rather than being loaded.

The absolute records produced consist of the relocated programs (including all relocatable subroutines), the linkage area and all referenced segments of the Basic Control System. These include:

Input/Output control subroutine (.IOC.)
All input/output equipment drivers
Memory Table (.MEM.)
System Linkage Area
Interrupt Processing area
Absolute location 2 and 3

In addition, the Loader Symbol Table, the common and linkage area bounds are punched in ASCII format on the end of the binary tape. Ten inches of feed frames separate the binary instructions and the ASCII data. This feature provides a record of the memory allocation.

At the completion of the "loading" process the Loader types the message "END".

To execute the program, it must be loaded using the Basic Binary Loader. To initiate execution, set 000002 into the P-Register and press RUN. The Loader has stored the transfer address of the program in locations 2 and 3 as follows:

2 contains JMP 3, I
3 contains <transfer address>

Separation of List and Binary Output

If the absolute binary output option is selected and the Teleprinter is used as both a list and punch device, the Loader halts before and after each line is printed to avoid punching the line and altering the binary output.

The halts and related procedures are as follows:

T-Register Contents	Explanation	Action
102055	A line is about to be printed.	Turn punch unit OFF. Press RUN.
102056	A line has been printed.	Turn punch unit ON. Press RUN.

INPUT/OUTPUT DRIVERS 4

4.1 GENERAL DESCRIPTION

An I/O driver, operating in the BCS environment, is responsible for controlling all data transfer between an I/O device and the cpu. It operates under control from the program.IOC. Its operating parameters are the user I/O request and the information contained in the device associated Equipment Table entry.

4.2 STRUCTURE

An I/O driver is a relocatable program segmented into two closed subroutines, termed the "initiator" and "continuator" sections. The entry point names for these two sections must be "D.nn" and "I.nn", respectively. The numeric value "nn" in the names is the equipment type code assigned to the device. For example, D.ØØ and I.ØØ are the entry points for the teleprinter driver; "ØØ" is the equipment type code assigned to a teleprinter.

```
                    NAM DRIVER D. nn

                 D. nn
                 ____________________
                 ____________________
Initiator Section ____________________
                 ____________________
                 ____________________
                 ____________________

                 I. nn
                 ____________________
                 ____________________
Continuator Section ____________________
                 ____________________
                 ____________________
                 ____________________
```

4.2.1 INITIATOR SECTION

This section is called directly from IOC with calling parameters including the address of the second word of the user I/O request and the address of the EQT entry for the referenced device. IOC sets these parameters in A and B and performs a JSB to the entry point "D. nn". Return to IOC from this section must be indirectly through D. nn.

On entry to D. nn,

(A) = Address of word 1 of 4-word EQT entry
(B) = Address of word 2 of I/O request

The initiator section of any driver must perform the functions described below.

1) Reject the IOC request and return to IOC (see step 6) if any of the following conditions exist:

   a. the driver is busy operating another device

   b. the referenced device is busy or inoperable

   c. the user request code or other parameters illegal for the device

   d. a DMA channel is not available and DMA is required for data transfer.

2) Extract the parameters from the user I/O request and save them within the driver storage.

3) Configure all I/O instructions in the driver to include the channel number for the reference device.

4) Indicate equipment in operation:

   a. set the "a" field in the EQT entry to 2 (busy) for the device called

   b. set an internal driver "busy" flag for the driver

   c. set a "busy" flag in IOC if a DMA channel is used

(To set a DMA flag in IOC:

Within the IOC program the two entry points DMAC 1, DMAC 2 contain the DMA channel locations (6 and 7 or 7 and 6). The signbit of the channel used must be set to 1 to indicate that the channel is busy.)

5) Initialize operating conditions and activate the device.

6) Return to IOC with the A and B registers set to indicate initiation or rejection and the cause of the reject:

(A) = Ø, operation initiated
= 1, operation rejected - reason in B-register

(B) = 1ØØØØØ, the device is busy or inoperable, or the driver is busy
= ØØØØØ1, a DMA channel is required but no channel is available
= ØØØØØØ, the request code or sub-function is not legal for the device

4.2.2 CONTINUATOR SECTION

This section is entered by device interrupt to continue or complete an operation. It may also be called from the Initiator Section to begin an operation. The entry point to this section is I. nn. There are no parameters on entry.

The continuator section of any driver must perform the functions described below.

1) Save all registers which will be used by the continuator section.

2) Perform the input or output of the next data item. If the transfer is not completed, restore the "saved" register and return control to the program.

NOTE: A driver for a device which inputs or outputs data independent of program control such as DMA would not include step 2. The device is activated by the initiator section (step 5), and the data transfer is immediately accomplished. The continuator section for such drivers merely completes the input or output operation.

3) When data transfer is completed (end-of-operation) or if a device malfunction is detected, set the following information in the EQT entry:

    The number of words or characters transferred (corresponding to the request) is set as a positive value in word 3. Bit 15 of word 3 is set to Ø or 1 to indicate the mode of transfer.

    The device status, actual or simulated, is set in bits 07-00 of word 2 and the "a" field (bits 15-14) in word 2 set to:

    0 - device available (not busy)
    1 - device available; the operation is complete but an error has been detected

    Bits 13-08 of word 2 must not be altered.

4) Clear all "busy" indicators. Clear the driver busy flag. If a DMA channel was used clear the flag in IOC.

5) Restore all registers saved at the entry.

6) Return indirectly through the entry point I. nn, with the following exception:

    If end-of-operation occurs for an output or function request, the driver returns to the entry point ".BUFR" in .IOC. This enables the buffered version of .IOC. to perform the automatic output buffering function. The standard version of .IOC. at this entry point just performs a normal return to the point of interruption. The calling sequence to .BUFR is:

```
         EXT    .BUFR
(P)      JSB    .BUFR
(P+1)    NOP          (holds return address from I. nn)
(P+2)    NOP          (holds EQT entry address)
```

PREPARE CONTROL SYSTEM 5

The Prepare Control System (PCS) program processes relocatable modules of the Basic Control System and produces an absolute version designed to work on a specific hardware configuration. It creates operating units of the Input/Output Control subroutine (.IOC.), the equipment driver subroutines, and the Relocating Loader. It also establishes the contents of certain locations used in interrupt handling. Options are available to define the equipment driver modules and other BCS system subroutines as relocatable programs to be loaded with the user's object program.

The Prepare Control System is an absolute program which is loaded by the Basic Binary Loader. It operates on a minimum configuration of 4K memory and a 2752A teleprinter. However, if a Paper Tape Reader and a Paper Tape Punch are available, the Prepare Control System will utilize these devices. PCS requests their assignment during the initialization phase.

After the Initialization phase is completed, each module of BCS is loaded and processed by PCS. The order in which the modules are processed is not significant except that the BCS Loader must be the last module loaded. Two modules, the Input/Output Control subroutine and the Loader, require that parameters be entered via the Keyboard Input unit after being loaded.

5.1 INITIALIZATION PHASE

During the Initialization phase, the system requests the channel assignments of the Paper Tape Reader and the Tape Punch if available. The operator supplies this information. Next the system requests the first and last words of available memory. The first word is the location in the base page following the locations required for interrupt processing (the interrupt locations and the locations containing the addresses of the Interrupt Processors). This location defines the start of the BCS system linkage area. The last word of available memory is usually the location prior to the protected area (e.g., 7677 for 4K memory, 17677 for 8K memory).

Example:

```
HS INP?                  message
10                       reply
HS PUN?                  message
11                       reply
FWA MEM?                 message
30                       reply
LWA MEM?                 message
17677                    reply
   .
   .
   .
```

5.2 LOADING OF BCS MODULES

After the initialization phase is completed, the system types "LOAD." The BCS modules are loaded using the Paper Tape Reader (if available) or the teleprinter. The modules may include .IOC., the equipment drivers, and the Relocating Loader. They can be loaded in any order provided that the Relocating Loader is last. The message is repeated after each module is loaded until the Loader has been processed. Diagnostics are printed if certain error conditions occur during the loading.

The absolute lower and upper bounds of each program within BCS are listed after the program is loaded. The format is as follows:

```
<program name>
 lllll   uuuuu
```

Equipment driver subroutines and interrupt processing sections that are to be used in relocatable form are identified during PCS processing but are not loaded. At the completion of the processing, PCS requests the missing subroutines. The proper response identifies each as external.

5.3 INPUT/OUTPUT EQUIPMENT PARAMETERS

After the Input/Output Control module is loaded, PCS requests the information needed to construct the Equipment Table (EQT) and Standard Equipment Table (SQT).†

Equipment Table Statements

PCS first types the messages "TABLE ENTRY" and "EQT". The operator responds by supplying the Equipment Table entries in the following format:

nn, D.ee [,D] [,Uu]

nn The channel number (select code) for the device. For a device connected to two or more channels, nn is the lower numbered channel.

D.ee The Basic Control System symbolic name for the related equipment driver subroutine. ee is the equipment type code used by BCS. Driver names are as follows:

D.00 — 2752A Teleprinter
D.01 — 2737A Punched Tape Reader
D.02 — 2753A Tape Punch
D.15 — Mark Sense Reader
D.20 — Kennedy 1406 Incremental Tape Transport
D.21 — 2020 Magnetic Tape Unit
D-22 — 3030 Magnetic Tape Unit
D-40 — Data Source Interface
D.41 — Integrating Digital Voltmeter
D.42 — Guarded Crossbar Scanner
D.43 — Time Base Generator

D A Direct Memory Access channel is required to operate the device.

Uu The physical unit number u (0-7) for addressing the device if it is attached to a multi-unit controller.

The same response is used regardless of whether the related subroutine driver is to be relocatable or absolute (part of BCS). If the driver is not encountered during processing, PCS prints the following:

```
I/O DRIVER?
D.EE
```

A response of "!" indicates that the driver is to be in relocatable form. (Any other response at this time is an error.) Drivers which use DMA or reference IOERR in the .IOC. may not be used externally.

The order in which the EQT statements are submitted defines the position of the entry in the Equipment Table. It also establishes the unit-reference number that the programmer uses in writing input/output requests to .IOC. The first statement entered describes the unit which is to be referenced as number 7_8; the second statement, number 10_8; the third statement, number 11_8; etc. Numbers 1 through 6 are reserved for Standard unit definition in the Standard Equipment Table.

The statement "/E" is entered to terminate the EQT input.

Example:

		Unit-Reference Number
*TABLE ENTRY	Message	
EQT?	Message	
10,D.01	Statement	7
11,D.02	Statement	10
12,D.00	Statement	11
/E	Terminator	

Standard Equipment Table Statements

In constructing the Standard Equipment Table, PCS types a mnemonic for the Standard unit and waits for the reply. The reply consists of the unit-reference number for a device previously described in the Equipment Table.

Example:

SQT?	message	
-KYBD?	message to assign Keyboard Input	
11 -TTY?	reply:	unit-reference number for Teleprinter message to assign Teleprinter Output
11 -LIB?	reply:	unit-reference number for teleprinter message to assign Relocatable Library

7	reply: unit-reference number for Punched Tape Reader
-PUNCH?	message to assign Punch Output
10	reply: unit-reference number for Tape Punch
-INPUT?	message to assign Input
7	reply: unit-reference number for Punched Tape Reader
-LIST?	message to assign List Output
11	reply: unit-reference number for Teleprinter

Direct Memory Access Statement

After the equipment tables are completed, PCS requests information about the availability of DMA channels to be controlled by the Input/Output Control and equipment driver subroutines. PCS types the message "DMA?" and the operator responds with the available DMA channel numbers. The format of the reply is:

$$c_1 \, [, c_2]$$

c_1 is 6 if one channel is available

c_2 is 7 if the second channel is available

If no DMA channel is available, the reply is 0 (zero).

Example:

DMA?	message
6,7	reply for two channels

If the reply contains any characters other than 0, 6 or 7, it is an error and a diagnostic is issued.

5.4 INTERRUPT LINKAGE PARAMETERS

After the Relocating Loader is loaded, PCS requests the parameters needed to set the Interrupt Linkage for Input/Output processing. The information required for each device includes:

> The interrupt location within the Reserved Location area in low core.
>
> The entry point name of the interrupt processing section in the equipment driver subroutine for the device.
>
> The address of the word in the Base Page which is to contain the 15-bit absolute address of this entry point name.

The same response is used regardless of whether the subroutine driver is to be relocatable or absolute (part of BCS). If the entry point was not encountered during processing, PCS prints the following:

```
*UN NAME
```

A response of ! indicates that the driver is to be in relocatable form. (Any other response at this time redefines the linkage.) Drivers which use DMA or reference IOERR in .IOC. cannot be used externally.

Given this information, PCS sets in the interrupt location a Jump Subroutine (Indirect) to the word holding the absolute address for the entry point of the Interrupt Processor.

Location	Content
10	JSB 20B,I
.	.
.	.
.	.
20	DEF I.01

10 is the interrupt location

20 holds the address of the entry point, I.01, of the Interrupt Processor.

PCS types the message "INTERRUPT LINKAGE?" The operator responds with a message in the following format:

$$a_1, a_2, \text{I.ee}$$

a_1 The address in low core of the interrupt location for the device (channel).

a_2 The address in the Base Page of the word to contain the absolute address of the Interrupt Processor entry point.

I.ee The entry point name of the Interrupt Processor section of the equipment driver subroutine. ee is the equipment type code used by BCS. Entry point names are as follows:

I.00 — 2752A Teleprinter
I.01 — 2737A Punched Tape Reader
I.02 — 2753A Tape Punch
I.15 — Mark Sense Reader
I.20 — Kennedy 1406 Incremental Tape Transport
I.21 and C.21† — 2020 Magnetic Tape Unit
I.22 and C.22† — 3030 Magnetic Tape Unit
I.43 — Time Base Generator

The statement "/E" is entered to terminate the Interrupt Linkage parameter input.

Example:

```
INTERRUPT LINKAGE?
```

message

```
10,20,I.01
```

reply: The Paper Tape Reader uses interrupt location 10. The absolute address for entry point I.01 is location 20 in the base page.

```
11,21,I.02
```

reply: The Tape Punch uses interrupt location 11. The address of I.02 is at location 21.

† Both the magnetic tape systems are connected to two channels; the lower numbered channel transfers data (I.21, I.22); the higher numbered channel transfers commands (C.21, C.22).

12,22,I,ØØ	The teleprinter uses interrupt location 12. The address of I.00 is at location 22.
/E	Terminates linkage parameters.

The response to the "INTERRUPT LINKAGE?" message may have the following form if a constant, for example a halt, is to be set in the interrupt location.

a, c

a The address in low core of the interrupt location for the device (channel).

c The constant in octal form that is to be stored at location a.

Example:

INTERRUPT LINKAGE?	message	
27,1Ø2Ø27	reply:	A halt executed when interrupt occurs on channel 27.
26,Ø	reply:	A NOP is executed when interrupt occurs on channel 26; the program resumes normal execution.

5.5 PROCESSING COMPLETION

When the Interrupt Linkage parameters have been supplied, PCS performs the following functions:

1. Prints the message *UNDEFINED SYMBOL followed by the entry point names of all system subroutines which have been referenced as externals but not loaded. At this point, PCS may continue and

the missing subroutines loaded or, the symbols may be added to the Relocating Loader's Loader Symbol Table. Undefined symbols are assigned as value of 77777 for an absolute address.

2. Completes the construction of the Loader Symbol Table.

3. Sets the Memory Table (symbolic location .MEM.) in the Relocating Loader to reflect the final bounds of available memory.

Following this, PCS prints a list of all Basic Control System entry points and the bounds of the System Linkage area in the Base Page.

Example:

```
.SQT.    17472
.EQT.    17500
.IOC.    17515
DMAC1    17676
DMAC2    17677
IOERR    17656
XSQT     17674
XEQT     17675
D.00     16745
I.00     17107
D.01     16406
I.01     16521
D.02     16115
I.02     16226
.LDR.    15413
HALT     16110
.MEM.    16110
LST      14102


*SYSTEM LINK
00030  00071
```

The final step in PCS processing is the punching of an absolute binary tape of the configured Basic Control System. This tape can be loaded by the Basic Binary Loader. When the tape is to be punched, BCS types the message *BCS ABSOLUTE OUTPUT. At the completion of the PCS run, the message *END is typed. The tape is punched using the tape punch unit, if available, or the teleprinter.

DEBUGGING SYSTEM 6

The debugging routine for BCS provides the following programming aids:

Print (dump) selected areas of memory in octal or ASCII format
Trace portions of the program during execution
Modify the contents of selected areas in memory
Modify simulated computer registers
Instruction and operand breakpoint halts
Initiate execution at any point in program
Debugging routine restart
Specifying relocatable program base

The Debugging routine supervises the operation of a program in the check-out (debugging) phase through the use of an interpretive mode of execution with simulated A, B, E overflow and P registers.

The Debugging routine is a relocatable program. It is loaded into memory after the user's relocatable programs and before the library subroutines are loaded. The Debugging routine makes use of the input/output control subroutine, IOC.

6.1 OPERATOR COMMUNICATION

All communication between the debugging routine and the user is done through the standard keyboard input and standard teleprinter output units normally assigned to a teleprinter.

After the program is loaded, the Debugging routine pauses to allow the first type-in. The operator then types one or more control statements to direct the operation of the Debugging routine. Each statement must be terminated by an end-of-statement mark which consists of a carriage return, CR , and a line feed LF . The last statement of the set must be a Run statement.

When an operation requested by a control statement is completed, a pause occurs (except for the Trace operation). The operator may then continue by typing a Run statement, or he may enter new control statements. To regain control at any other time, the operator must use Switch 15. Caution must be used, however, when input/output operations are in progress; setting the switch causes a message to be typed. This action may disrupt any incomplete I/O operation.

6.2 CONTROL STATEMENTS

The basic format of the control statement is a single alphabetic character representing the requested operation followed by a parameter list containing the arguments for the operation separated by commas. The statement is of variable length and is terminated by (CR) (LF). The numeric fields in the parameter list must be in octal; leading zeros may be omitted.

Program Relocation Base

M, a

This statement defines the program relocation base, a, as the absolute origin in memory of the user's relocatable program. This address may be obtained from the listing produced by the Relocating Loader during loading. If not specified, a value of zero is assumed. The value is added to all address parameters entered by the operator.

Specification of this value allows subsequent reference in the control statements to addresses as shown on the program listing produced by the Assembler or the FORTRAN compiler. If this control statement is not used, program address parameters for other control statements must be absolute. DEBUG does not check for memory address greater than the core size; therefore, locations in the base page may be altered if the program relocation base is too high.

Example:

M, 2ØØØ

Set Memory

S, a, v_1, $v_2, \ldots, v_n$

The above statement allows the user to set one or more values into locations defined by the first address, a. The value specified for v_1 is stored in location a; the value for v_2, in location a + 1; and so forth. To specify that an existing value in memory is to remain unchanged, two consecutive commas are used in the control statement. Any number of values may be entered via one control statement provided the length of the statement does not exceed 72 characters.

Example:

```
    S,5,062006
    S,30,136100,026040
    S,40,136101,026050
```

Set Register

```
    W,r,v
```

This statement sets the value, v, into register, r, where the register is defined as follows:

```
    r = A,A-Register
      = B,B-Register
      = E,E-Register
      = O,Overflow
```

Since the Debugging routine simulates the register, the results of a Set Register operation are not reflected on the computer front panel.

Examples:

```
    W,B,2
    W,A,102000
    W,E,1
```

Dump Memory

D, A, a_1, a_2

D, B, a_1, a_2

The second parameter indicates the format of the print-out: A specifies ASCII, B specifies octal. The address a_1 designates the location of the word or the first of a series of words that is to be dumped. If the second address, a_2, is greater than a_1, a block of memory, a_1 through a_2, is printed. If a_2 is the same as a_1, only one location is printed.

After the data is printed, the Debugging routine waits for the operator to enter another control statement.

Example:

```
    D,A,430,477
```

Breakpoint Halt

B,I,a
B,O,a

The first form specifies the address, a, of an instruction breakpoint. Before the instruction at address a is executed, the Debugging routine writes a standard breakpoint message (See Output Formats).

The second form specifies the address, a, of an operand breakpoint. When the Debugging routine detects an effective operand address equal to the value of a, it writes a standard breakpoint message. The operand breakpoint occurs before the memory reference is completed and the register contents in the message are the contents during the instruction execution and not at completion.

After the breakpoint message is transmitted, the Debugging routine waits for the user to enter another control statement.

One or both types of breakpoint halts may be selected. Once selected, a breakpoint address remains in effect until a new address is selected, until a Restart statement is entered, or until the selection is terminated by the statements:

Trace

B,I,Ø or B,O,Ø

T, a_1[, a_2]

When the Trace operation is specified, the execution of the instruction located at address a_1, or the execution of every instruction within the area a_1 through a_2, causes the printing of a standard breakpoint message. (See "Output Formats.") The printing occurs before each instruction is executed. Each time the a_1 - a_2 area is reached, the printing resumes; no pause occurs on completion as in the other debugging routine operations.

The area to be traced must not contain calls to the input/output control routine, IOC. The Trace operation uses IOC to print the breakpoint message. An attempt to trace I/O operations will result in I/O errors.

The trace of the area remains in effect until a new area is selected or until the selection is terminated by the statement:

T,Ø

To enter a new Trace control statement while the program is in operation, Switch 15 must be used.

Run

R [,a]

This statement is used to initiate the execution of the program being debugged. It can also be used to continue execution after a pause in execution (caused by setting switch register bit 15 to 1 or by breakpoint halt). If the letter R only is entered, execution starts with the next sequential instruction in the user's program. To start at another location, the operator enters the address, a.

Restart

A

This statement, consisting of the letter "A" is used to abort the current operation and restart. This results in all debugging routine and input/output operations in progress being cleared.

6.3 CONTROL STATEMENT ERROR

If an incorrect control statement is entered, the following message is typed:

"ENTRY ERROR"

This indicates that the character representing the operation is invalid, or that an illegal parameter has been typed. To recover, type in the correct control statement.

6.4 HALT

Any halt operations coded within the user's program result in a typeout consisting of the letter H followed by the standard breakpoint message. The operator can then type in one or more control statements or can reinitiate program execution (with the R control statement).

6.5 INDIRECT LOOP

The debugging routine counts levels when indirect addressing is detected. When ten consecutive levels of indirect addressing have occurred, an indirect address loop is assumed and the following is typed out:

> "INDIRECT LOOP"
>
> L <standard breakpoint message>

6.6 OUTPUT FORMATS

The Debugging routine operations may produce either of two printed outputs: the standard breakpoint message and the memory dump.

Standard Breakpoint Message

Each standard breakpoint message has the following format:

$$<id>P = v_1 I = v_2 A = v_3 B = v_4 E = v_5 O = v_6 MA = v_7 \; MC = v_8$$

The <id> is a letter identifying the operation producing the output:

id = I, Instruction breakpoint
 = O, Operand breakpoint
 = T, Trace
 = S, Switch 15 set up
 = L, Indirect Loop
 = H, Halt in object program

The v's are octal values of registers and memory locations as follows:

P - P-Register (instruction address)
I - Instruction (contents)
A - A-Register
B - B-Register
E - E-Register
O - Overflow
MA - Effective operand address of a memory reference instruction
MC - Contents of effective address of a memory reference instruction

Dump

The Dump output record format consists of the contents up to 8 consecutive words preceded by the address of the first word:

	addr.	word$_1$	word$_2$ . . .	word$_8$
Octal:	aaaaa	oooooo	oooooo . . .	oooooo
ASCII:	aaaaa	cc	cc	cc

Octal words consist of 6 octal digits; ASCII words are listed as two ASCII characters. The contents of eight or more consecutive words are not written or they are the same as the last word of the previous record. Instead, a record containing only an asterisk is produced.

6.7 OPERATING PROCEDURES

The following procedures indicate the sequence of steps for use of the Debugging routine.

A. Set the Teleprinter to LINE and check that all equipment to be used is operable.

B. Load Basic Control System using the Basic Binary Loader.

C. Set a starting address of 2 and zero the Switch Register.

D. Establish Relocating Loader parameters. (If relocation base is to be entered during operation of the debugging routine, the address must be obtained during loading by setting Switch 15 to 0.)

E. Load user relocatable object programs.

F. Load Debugging program (treated as a relocatable program). †

G. Load Relocatable Library routines.

† The Debugging routine need not be loaded as the last relocatable program. If loaded in any other order, however, the absolute address assigned to the symbolic location DEBUG must be entered manually as the starting address for the program.

H. Press RUN.

I. The program pauses to allow the operator to type in the control statements.

J. The program may be restarted at any point by entering the absolute address assigned to the symbolic location DEBRS into the P-Register, and pressing RUN.

6.8 EXAMPLE

The routine employed in this example is a simple loop which totals the contents of a block of data. In order to imbue it with a practical aspect, assume that program "TOTAL" computes personal expenses for a 31-day month. Data (each day's expenses) is read in from the Punched Tape Reader. The sum is printed out on the Teleprinter.

The program is written and assembled as below. To check it out a data tape, consisting of a series of 10's is prepared:

```
10    CR LF
10    CR LF
10    CR LF
 .
 .
 .
```

```
PAGE 0002 #01


0001  00000                NAM TOTAL
0002  00000 000000  START  NOP
0003  00001 062162R        LDA =D-31
0004  00002 072156R        STA CTR
0005  00003 062163R        LDA =B5
0006  00004 006404         CLB,INB
0007  00005 016004X        JSB .DIO.
0008  00006 000000         ABS 0
0009  00007 000014R        DEF *+5
0010  00010 016006X        JSB .IOR.
0011  00011 016001X        DST INPUT,I
      00012 100055R
0012  00013 016005X        JSB .RAR.
0013  00014 066055R        LDB INPUT               INPUT THE DATA
0014  00015 046164R        ADB =B2
0015  00016 076055R        STB INPUT
0016  00017 036156R        ISZ CTR
0017  00020 026003R        JMP START+3
```

```
0019  00021 062162R        LDA =D-31
0020  00022 072156R        STA CTR          INITIALIZE
0021  00023 016002X        DLD =F0.0
      00024 000165R
0022  00025 016001X        DST ANSW
      00026 000154R

0024  00027 016002X        DLD .MON,I
      00030 100054R
0025  00031 016003X        FAD ANSW
      00032 000154R
0026  00033 016001X        DST ANSW
      00034 000154R
0027  00035 066054R SUM    LDB .MON
0028  00036 046164R        ADB =B2          ADDITION LOOP
0029  00037 076054R        STB .MON
0030  00040 036156R        ISZ CTR
0031  00041 026035R        JMP SUM

0033  00042 062164R        LDA =B2
0034  00043 006400         CLB
0035  00044 016004X        JSB .DIO.
0036  00045 000157R        DEF OUTPT
0037  00046 000053R        DEF *+5
0038  00047 016002X        DLD ANSW         OUTPUT THE RESULT
      00050 000154R
0039  00051 016006X        JSB .IOR.
0040  00052 016007X        JSB .DTA.
0041  00053 102077         HLT 77B

0043  00054 000056R .MON   DEF MONTH
0044  00055 000056R INPUT  DEF MONTH
0045  00056 000000  MONTH  BSS 62
0046  00154 000000  ANSW   BSS 2
0047  00156 000000  CTR    BSS 1
0048                       EXT .DIO.,.RAR.,.IOR.,.DTA.
0049  00157 024106  OUTPT  ASC 3,(F8.2)
      00160 034056
      00161 031051


      00162 177741
      00163 000005
      00164 000002
      00165 000000
      00166 000000
0050                       END START
**  NO ERRORS*
```

The "TOTAL" object tape is loaded by the Basic Control System. The debugging system is loaded next and then the library tape. The program is executed using the Debugging System by the following instructions:

```
M,2000      Set program relocation base

B,I,53      Breakpoint instruction is 53, the location of the terminating
            halt in the program.

R,1         Initiate execution at statement 1
```

```
10.00
H P=00053   I=102077   A=177777   B=006115   E=0   O=1
```

The correct answer for the test data would be "31.00", not the 10.00 that was output.

The procedure below illustrates one method for detecting errors in the program.

```
M,2000      Set program relocation base
```

Dump a portion of the storage area MONTH

```
D,B,56,70
DUMP--BASE = 02000

00056                                           050503 000333
00060 001253 000000 000000 004267 017700 000000 053070 011770
00070 002256
```

Read in the data:

```
B,I,21
R,1
I P= 00021 I=062162 A=000000 B=002154 E=0 O=0 MA=00162 MC=177741
```

Check to see that the data has been stored in memory:

```
D,B,56,70
DUMP--BASE = 02000

00056                                           050000 000010

00060 050000 000010 050000 000010 050000 000010 050000 000010

00070 050000
```

Knowing that the data has been stored in MONTH, perform the first addition:

```
B,I,35
R,21
I P= 00035 I=Ø66Ø54 A=Ø5ØØØØ B=ØØØØ1Ø E=Ø O=Ø MA=ØØØ54 MC=ØØ2Ø56
```

Check to see that the first day's expenses have been stored at ANSW:

```
D,B,154,155
DUMP--BASE = Ø2ØØØ

 ØØ154                                          Ø5ØØØØ   ØØØØ1Ø
```

The first addition was executed. Perform the remaining additions by looping:

```
B,I,42
R,35
I P= ØØØ42 I=Ø62164 A=Ø5ØØØØ B=ØØ2154 E=Ø O=Ø MA=ØØ164 MC=ØØØØØ2
```

Check final total ~~in~~ ANSW.

```
D,B,154,155
DUMP--BASE = Ø2ØØØ

 ØØ154                                          Ø5ØØØØ   ØØØØ1Ø
```

Here, if not previously, the error should be detected; the program does not perform more than the first addition. The label sum has been placed in the wrong instruction. It should be in location 27 preceding the "DLD . MON, I" instruction.

HP CHARACTER SET A

ASCII CHARACTER FORMAT

b_7				0	0	0	0	1	1	1	1
b_6				0	0	1	1	0	0	1	1
b_5				0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1								
0	0	0	0	NULL	DC$_0$	ƀ	0	@	P	UNASSIGNED	UNASSIGNED
0	0	0	1	SOM	DC$_1$	!	1	A	Q		
0	0	1	0	EOA	DC$_2$	"	2	B	R		
0	0	1	1	EOM	DC$_3$	#	3	C	S		
0	1	0	0	EOT	DC4 (STOP)	$	4	D	T		
0	1	0	1	WRU	ERR	%	5	E	U		
0	1	1	0	RU	SYNC	&	6	F	V		
0	1	1	1	BELL	LEM	' (APOS)	7	G	W		
1	0	0	0	FE$_0$	S$_0$	(	8	H	X		
1	0	0	1	HT/SK	S$_1$	)	9	I	Y		
1	0	1	0	LF	S$_2$	*	:	J	Z		
1	0	1	1	V$_{TAB}$	S$_3$	+	;	K	[		
1	1	0	0	FF	S$_4$	, (COMMA)	<	L	\		ACK
1	1	0	1	CR	S$_5$	−	=	M	]		①
1	1	1	0	SO	S$_6$	.	>	N	↑		ESC
1	1	1	1	SI	S$_7$	/	?	O	←		DEL

Standard 7-bit set code positional order and notation are shown below with b_7 the high-order and b_1 the low-order, bit position.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
EXAMPLE: The code for "R" is:	1	0	1	0	0	1	0

LEGEND

NULL	Null/Idle
SOM	Start of message
EOA	End of address
EOM	End of message
EOT	End of transmission
WRU	"Who are you?"
RU	"Are you...?"
BELL	Audible signal
FE$_0$	Format effector
HT	Horizontal tabulation
SK	Skip (punched card)
LF	Line feed
V$_{TAB}$	Vertical tabulation
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DC$_0$	Device control reserved for data link escape

DC$_1$-DC$_3$	Device Control
DC$_4$(Stop)	Device control (stop)
ERR	Error
SYNC	Synchronous idle
LEM	Logical end of media
S$_0$-S$_7$	Separator (information)
ƀ	Word separator (space, normally non-printing)
<	Less than
>	Greater than
↑	Up arrow (Exponentiation)
←	Left arrow (Implies/Replaced by)
\	Reverse slant
ACK	Acknowledge
①	Unassigned control
ESC	Escape
DEL	Delete/Idle

BINARY CODED DECIMAL FORMAT

Kennedy 1406/1506 ASCII-BCD Conversion

Symbol	BCD (octal code)	ASCII Equivalent (octal code)
(Space)	2Ø	Ø4Ø
!	52	Ø41
#	13	Ø43
$	53	Ø44
%	34	Ø45
&	6Ø	Ø46
'	14	Ø47
(	34	Ø50
)	74	Ø51
*	54	Ø52
+	6Ø	Ø53
,	33	Ø54
-	4Ø	Ø55
.	73	Ø56
/	21	Ø57
Ø	12	Ø6Ø
1	Ø1	Ø61
2	Ø2	Ø62
3	Ø3	Ø63
4	Ø4	Ø64
5	Ø5	Ø65
6	Ø6	Ø66
7	Ø7	Ø67
8	1Ø	Ø7Ø
9	11	Ø71
:	15	Ø72
;	56	Ø73
<	76	Ø74
=	13	Ø75
>	16	Ø76
?	72	Ø77
@	14	1ØØ
A	61	1Ø1
B	62	1Ø2
C	63	1Ø3
D	64	1Ø4
E	65	1Ø5
F	66	1Ø6
G	67	1Ø7
H	7Ø	11Ø
I	71	111
J	41	112
K	42	113
L	43	114
M	44	115
N	45	116
O	46	117
P	47	12Ø
Q	50	121
R	51	122
S	22	123
T	23	124
U	24	125
V	25	126
W	26	127
X	27	13Ø
Y	30	131
Z	31	132
[	75	133
\	36	134
]	55	135

Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

HP 2020 ASCII — BCD Conversion

Symbol	ASCII (Octal code)	BCD (Octal code)
(Space)	4Ø	2Ø
!	41	52
"	42	37
#	43	13
$	44	53
%	45	34
&	46	60 †
'	47	36
(	5Ø	75
)	51	55
*	52	54
+	53	6Ø
,	54	33
-	55	4Ø
.	56	73
/	57	21
Ø	6Ø	12
1	61	Ø1
2	62	Ø2
3	63	Ø3
4	64	Ø4
5	65	Ø5
6	66	Ø6
7	67	Ø7
8	7Ø	1Ø
9	71	11
:	72	15
;	73	56
<	74	76
=	75	35
>	76	16
?	77	72
@	1ØØ	14
A	1Ø1	61
B	1Ø2	62
C	1Ø3	63
D	1Ø4	64
E	1Ø5	65
F	1Ø6	66
G	1Ø7	67
H	11Ø	70
I	111	71
J	112	41
K	113	42
L	114	43
M	115	44
N	116	45
O	117	46
P	12Ø	47
Q	121	50
R	122	51
S	123	22
T	124	23
U	125	24
V	126	25
W	127	26
X	13Ø	27
Y	131	30
Z	132	31
[	133	75 ‡
]	135	55 ‡
↑	136	77
←	137	32

† BCD code of 60 always converted to ASCII code 53 (+).

‡ BCD code of 75 always converted to ASCII code 50 (() and
BCD code of 55 always converted to ASCII code 51 ()).

HP 2761A-007 Mark Sense Card Reader

Data read from Mark Sense Cards is converted from the same Hollerith codes used for punched cards to ASCII codes (octal equivalents) and packed two characters per computer word. The first character and every other character after it are placed in the upper half (bits 15 thru 8) of successive words in the buffer. The second character and every other character after it are placed in the lower half (bits 7 thru 0) of those same successive words. Thus, each character has the potential of either one of two representations in a computer word, depending on its position within the reading sequence. Both of these potentials are listed for each character available from Mark Sense Cards in Table A-3, starting below.

For example, if the word HEMP were being read the ASCII octal equivalent for H as the first character is 044000, which is stored as

```
 15          87         0
| 0 100 100 000 000 000 |
```

Next, the ASCII octal equivalent for E as the second character is 000105, which is stored as

```
 15          87         0
| 0 100 100 001 000 101 |
```

The first packed computer word then, is

```
 15          87         0
| 0 100 100 001 000 101 |
```

Finally, the next two characters M (046400) and P (000120) are stored in the next packed computer word as

```
 15          87         0
| 0 100 110 101 010 000 |
```

CHARACTER CONVERSIONS-MARK SENSE CARD READER

Hollerith or ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000100
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
0	030000	000060
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065

CHARACTER CONVERSIONS-MARK SENSE CARD READER

Hollerith or ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
(space)	020000	000040
!	020400	000041
” (quote)	021000	000042
#	021400	000043
$	022000	000044
%	022400	000045
&	023000	000046
’ (apostrophe)	023400	000047
(	024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
, (comma)	026000	000054
– (hyphen or minus)	026400	000055
. (period)	027000	000056
/	027400	000057
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
c (cent) or [	055400	000133
(not mark) or]	056400	000135
| (vertical bar*) or ↑	057000	000136
_ (underscore**) or ←	057400	000137
0–8–2 or \	056000	000134

*NUMERIC Y

**NUMERIC W

EQUIPMENT TABLE B

The Equipment Table (EQT) provides information for the input/output control routine, .IOC., and the equipment driver subroutines. The table contains an entry for each peripheral device attached to a Computer configuration.

The table is constructed as a block of entries assembled by the Prepare Control System routine. The first word of the table, at the symbolic entry point .EQT., contains the number of entries in the table. An entry in the table is referenced according to its position. The numbers 1 through 6 are reserved for Standard units (see Standard Equipment Table). The number 7_8 appearing in a program refers to the 1st table entry; the number 10_8, the second, and so forth. The numbers may be in the range 7_8-74_8 with the largest value being determined by the number of units of equipment available at the installation.

The 4-word entry for each device contains the following information:

The channel number of the device (10_8-76_8)

A Direct Memory Access channel indicator if pertinent

Absolute address of equipment driver subroutine

Equipment type identification code.

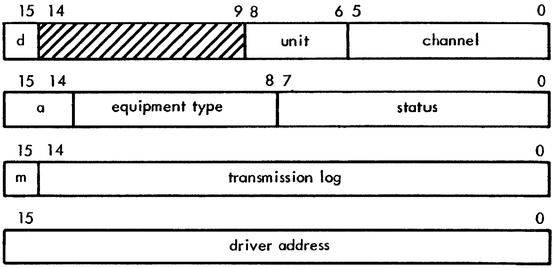
The above information is static for each installation; it is not altered by .IOC. The entry also contains dynamic information which is supplied by the equipment driver subroutine. This information includes:

Status of operation (i.e., in progress or complete)

Status of equipment

Number of characters or words transmitted when the operation is completed.

The format of the entry is as follows:



d = Direct Memory Access channel indicator

1 DMA channel is to be used for each data transmission operation

0 DMA channel not required

unit = Physical unit number (0-7) used to address the device if it is attached to a multi-unit controller.

channel = The channel number (select code) for the physical device (also the low core location containing a JSB to the related interrupt subroutine.)

a = Availability of device:

0 The device is available; the previous operation is complete.

1 The device is available; the previous operation is complete but a transmission error has been detected.

2 The device is not available for another request; the operation is in progress.

equipment type = This field contains a 6-bit code that identifies the device:

00-07 – Paper Tape devices
00 Teleprinter
01 Paper Tape Reader
02 Tape Punch

10-17 – Unit Record devices
15 Mark Sense Reader

20-37 – Magnetic Tape and Mass Storage devices
20 Kennedy 1406 Incremental Tape Transport
21 2020 Magnetic Tape Unit
22 3030 Magnetic Tape Unit

40-77 – Instrumentation devices
40 Data Source Interface
41 Integrating Digital Voltmeter
42 Guarded Crossbar Scanner
43 Time Base Generator

status = The status field indicates the actual status of the device when the data transmission is complete. The contents depend on the type of device (see Status Table).

m = This bit defines the mode of the data transmission:

0 ASCII or BCD

1 Binary

transmission log = This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in the calling sequence. The value is stored in this field only when the input/output request has been completed, therefore, when all data is transmitted or when a transmission error is detected.

driver address = Absolute address of the entry point for the associated driver subroutine for the device.

STATUS TABLE

Status Bit / Device	7	6	5	4	3	2	1	0
2752A Teleprinter			End of Input Tape					
2737A Punched Tape Reader			End of Tape					
2753A Tape Punch			Tape Supply Low					
Kennedy 1406 Incremental Tape Transport			End of Tape		BT			DB
2020A Magnetic Tape Unit	EOF	ST	End of Tape	TE	I/O R	NW	PA	DB
3030 Magnetic Tape Unit	EOF	ST	End of Tape	TE	I/O R	NW	PA	DB
Mark Sense Reader						PF	RNR	HE or SF

BT	=	Broken Tape
DB	=	Device Busy
EOF	=	End of File
ST	=	Start of Tape
TE	=	Timing Error
I/OR	=	I/O Reject
NW	=	No Write (write enable ring missing or tape unit is rewinding)
PA	=	Parity Error
HE	=	Hopper Empty
SF	=	Stacker Full
RNR	=	Reader Not Ready
PF	=	Pick Fail

STANDARD UNIT EQUIPMENT TABLE C

The Standard Unit Equipment Table (SQT) allows reference to input/output devices designated as Standard units. The Table contains six 1-word entries. Each entry corresponds to a particular Standard unit and contains a pointer to the Equipment Table. The Standard units are as follows:

Number	Name
1	Keyboard Input
2	Teleprinter Output
3	Program Library
4	Punch Output
5	Input
6	List Output

The number defines the position in the SQT at which the device is listed. Each Standard unit may be a different device, or a single physical device may represent several Standard units. The value of the pointer in the SQT is the position of the physical unit's entry in the EQT, with the lowest value being 7_8.

IOC WITH OUTPUT BUFFERING

IOC with Output Buffering is an extension of the standard version and provides for automatic stacking and buffering of all output and function requests. This involves moving an output request and associated buffer into available memory and adding the request location into a thread of stacked requests for the referenced unit. At the completion of an output operation, the next entry in the stack for the unit is initiated by IOC. The processing of output/function requests for a particular unit is according to the order of the requests (first in/first out). This version of IOC requires the use of the program MEMRY to perform the allocation and release of blocks of available memory. If available memory is exhausted when an allocation is attempted, IOC repeats the call until space is made available, i.e., previous blocks are released.

PRIORITY OUTPUT

A "priority" write or function request has been added for use with the Buffered version of .IOC. A priority request is processed immediately without the request and buffer being moved to available memory. The current operation in the stack is suspended, the priority request processed and the suspended operation re-initiated. The priority feature is useful for writing messages or diagnostics for immediate action or for performing output without reserving a segment of available memory for request/buffer storage. (All output performed by the BCS Relocating Loader is done as priority requests for the latter reason.) If two or more priority requests are called in immediate succession (without intervening status checks), the last requested operation is performed with the previous ones being "lost."

A "Priority" request (i.e., Write function) is indicated by setting bit Ø9 of Word 2 of the request call = 1. Bit Ø9 = 0 means normal operation with the Standard IOC and means the request will be stacked and buffered with the extended version.

Example: "Priority" Write to Teleprinter

```
JSB   .IOC.
OCT   21ØØ2
JMP   REJ
DEF   BUFFR
DEC   -37
```

OPERATING ENVIRONMENT

IOC with Output Buffering provides for writing a data block on more than one output device in parallel and does not restrict output rates to the lowest speed device. Because all requests and buffers are moved into available memory for subsequent processing, peak load output processing is not delayed due to device speed or saturated buffer storage within the bounds of user programs. System I/O saturation occurs when available memory is exhausted.

RESTRICTIONS

The routines used to allocate/release blocks in available memory and to initiate stacked output requests operate with the Interrupt System disabled. Therefore, the use of medium/high speed synchronous I/O devices (e.g., HP 2020 Magnetic Tape) under program control is not recommended because of possible data loss.

An I/O driver routine operating under the extended version of IOC may not be used to control more than one like device. This is because the buffering control routine in IOC only checks for stacked requests referencing the unit on which an operation was just completed.

HALT CONDITIONS

Irrecoverable error conditions are identical to the Standard version of IOC. The location of the halt is at the entry point "IOERR". These conditions are:

A-Register	B-Register	Meaning
Ø	Location at Request	Request Code Illegal
1	Location at Request	Unit Reference Illegal
Ø	Ø	Write request for an input only device.

I/O ERROR CONDITIONS

The routine .BUFR in the version of IOC with Output Buffering checks for error conditions of the end of each output operation. If any error conditions and End-of-Tape or Tape Supply Low, etc. conditions are present, IOC halts to allow the condition to be corrected. Processing is continued by pressing RUN.

Halt: (T) = 1Ø2Ø7Ø
(A) = Word 2 of EQT entry (Status word)
(B) = Hardware I/O address of unit

An addition has been made to this routine to handle requests for buffered output of records too long to be buffered with the amount of memory available. If such a request is made, the following occurs:

a. IOC outputs the contents of any buffers which have been previously "stacked" for the referenced I/O device.

b. The computer halts to inform the user that his program cannot buffer output records of the length requested. The contents of the registers are as follows:

(T) = 102001
(A) = Maximum length record that can be buffered with the amount of memory available.
(B) = Memory location of the output request which caused the halt.

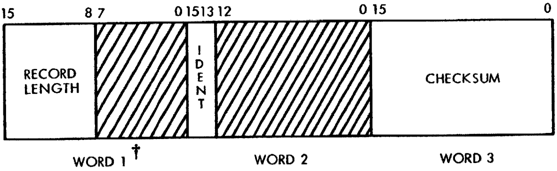
The user restarts the program by pushing the RUN button. The output request is honored immediately without buffering. IOC waits until the output operation is complete before returning control to the program. This ensures that the data area is not modified before the complete record is output, and that the output results are identical to those produced if buffered output of the record had occurred.

RELOCATABLE TAPE FORMAT

NAM RECORD

CONTENT

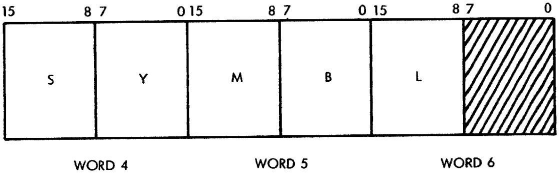
EXPLANATION



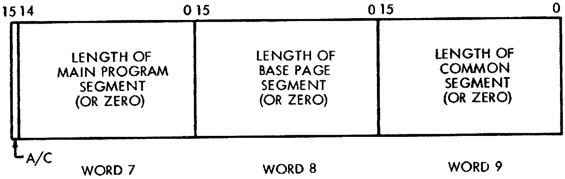
RECORD LENGTH = 9 WORDS

IDENT = 001

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS IN RECORD EXCLUDING WORDS 1 AND 3.

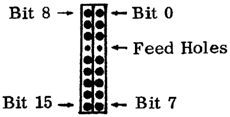


SYMBL: FIVE CHARACTER NAME OF PROGRAM

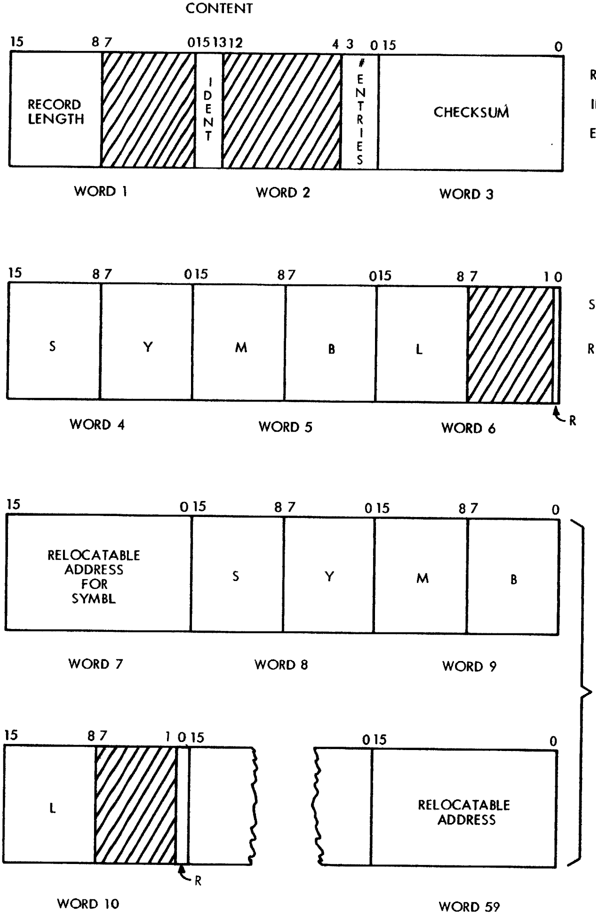

15 14
0 15
0 15
0
LENGTH OF MAIN PROGRAM SEGMENT (OR ZERO)
LENGTH OF BASE PAGE SEGMENT (OR ZERO)
LENGTH OF COMMON SEGMENT (OR ZERO)
A/C
WORD 7
WORD 8
WORD 9

A/C: BINARY TAPE PROCESSOR
= 0 IF ASSEMBLER PRODUCED
= 1 IF COMPILER PRODUCED

†Each word represents two frames arranged as follows:



ENT RECORD



EXPLANATION

RECORD LENGTH = 7-59 WORDS

IDENT = 010

ENTRIES: 1 to 14 ENTRIES PER PROGRAM; EACH ENTRY IS FOUR WORDS LONG.

SYMBL: 5 CHARACTER ENTRY POINT SYMBOL

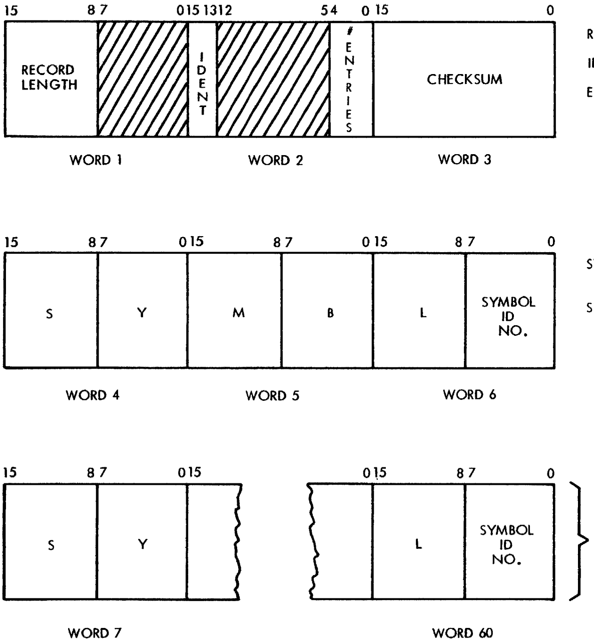
R: RELOCATION INDICATOR
= 0 IF PROGRAM RELOCATABLE
= 1 IF BASE PAGE RELOCATABLE

WORDS 4 THROUGH 7 ARE REPEATED FOR EACH ENTRY POINT SYMBOL.

EXT RECORD

CONTENT

EXPLANATION



RECORD LENGTH = 6-60 WORDS

IDENT = 100

ENTRIES: 1 TO 19 PER RECORD; EACH ENTRY IS THREE WORDS LONG

SYMBL: 5 CHARACTER EXTERNAL SYMBOL

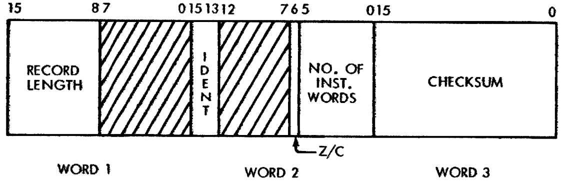
SYMBOL ID. NO.: NUMBER ASSIGNED TO SYMBL FOR USE IN LOCATING REFERENCE IN BODY OF PROGRAM.

WORDS 4 THROUGH 6 REPEATED FOR EACH EXTERNAL SYMBOL (MAXIMUM OF 19 PER RECORD).

DBL RECORD

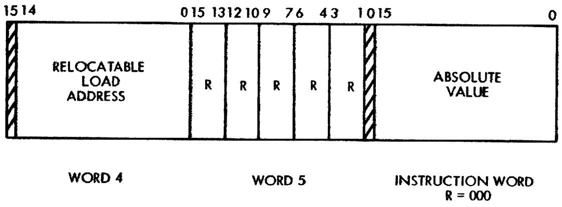
CONTENT

EXPLANATION



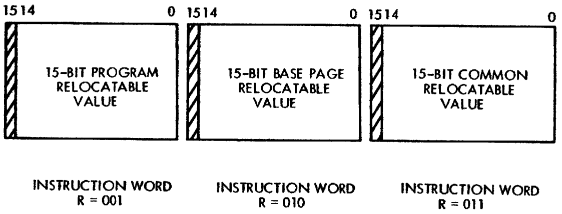
RECORD LENGTH = 5-60 WORDS
IDENT = 011
Z/C: BASE/CURRENT PAGE LOADING
= 0 FOR BASE PAGE
= 1 FOR CURRENT PAGE

NO. OF INST. WORDS: 1 TO 45 LOADABLE INSTRUCTION WORDS PER RECORD



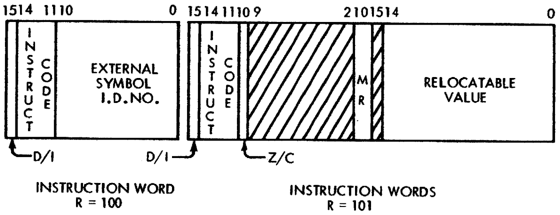
RELOCATABLE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW.

R's: RELOCATION INDICATORS:
000 = ABSOLUTE
001 = 15-BIT PROGRAM RELOCATABLE
010 = 15-BIT BASE PAGE RELOCATABLE
011 = 15-BIT COMMON RELOCATABLE
100 = EXTERNAL REFERENCE
101 = MEMORY REFERENCE


15 14
0
15 14
0
15 14
0
15-BIT PROGRAM RELOCATABLE VALUE
15-BIT BASE PAGE RELOCATABLE VALUE
15-BIT COMMON RELOCATABLE VALUE
INSTRUCTION WORD
R = 001
INSTRUCTION WORD
R = 010
INSTRUCTION WORD
R = 011

R_1 IS RELOCATION INDICATOR FOR INSTRUCTION $WORD_1$; R_2, FOR INSTRUCTION $WORD_2$; ETC-MEMORY REFERENCE INSTRUCTIONS USE TWO WORDS, WITHIN THE TWO-WORD GROUP, "MR" INDICATES RELOCATABILITY OF OPERAND SPECIFIED IN SECOND WORD:

00 = PROGRAM RELOCATABLE
01 = BASE PAGE RELOCATABLE
10 = COMMON RELOCATABLE



D/I: INDIRECT ADDRESSING

0 = DIRECT
1 = INDIRECT

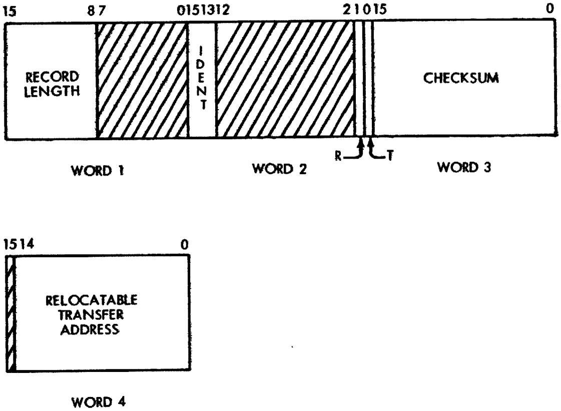
Z/C: BASE/CURRENT PAGE LOCATION OF OPERAND ADDRESS AS DETERMINED BY LOADER.

0 = BASE PAGE
1 - CURRENT PAGE

END RECORD

CONTENT

EXPLANATION



RECORD LENGTH = 4 WORDS

IDENT = 101

R: RELOCATION INDICATOR FOR TRANSFER ADDRESS

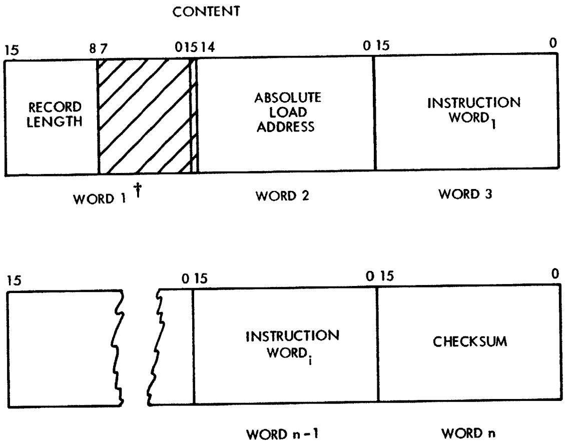
= 0 IF PROGRAM RELOCATABLE
= 1 IF BASE PAGE RELOCATABLE

T: TRANSFER ADDRESS INDICATOR

= 0 IF NO TRANSFER ADDRESS IN RECORD

= 1 IF TRANSFER ADDRESS PRESENT

ABSOLUTE TAPE FORMAT



EXPLANATION

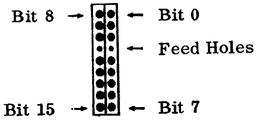
RECORD LENGTH = NUMBER OF WORDS IN RECORD EXCLUDING WORDS 1 AND 2 AND THE LAST WORD.

ABSOLUTE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW

INSTRUCTION WORDS: ABSOLUTE INSTRUCTIONS OR DATA

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS EXCEPT FIRST AND LAST

†Each word represents two frames arranged as follows:



HOW TO GENERATE A BASIC CONTROL SYSTEM G

The stand-alone program Prepare Control System (PCS) is used to generate BCS. The following parameters must be specified during generation (all numbers typed in octal):

First Word of Available Memory (FWA MEM)

This is the lowest memory location that is available to PCS for BCS construction. It should be higher than the last linkage location used in the Interrupt Table and if the BCS is to be used within MTS (Magnetic Tape System) it must be set to exactly 110_8 (to allow for MTS linkage locations). (Interrupt Table must be pre-planned before running PCS, since (FWA MEM) depends upon Interrupt Table length.)

Last Word of Available Memory (LWA MEM)

This is the highest memory location available to BCS. This value depends on the core size and the context as follows:

Core Size	Last Word BCS	BCS in MTS
4K	7677	
8K	17677	15777
16K	37677	35777
24K	57677	
32K	77677	

Equipment Table (EQT)

A table of varying size whose entries are numbered sequentially starting with 7. The user relates each entry to a specific I/O device and to an I/O driver. There must be at least one EQT entry per device to be used in BCS.

Standard Unit Table (SQT)

A set of 6 numbers (chosen from the EQT) that specify devices for standard functions (i.e., keyboard, list output, etc.).

Interrupt Table

The set of memory locations where interrupts may occur and a matching set of linkage locations (one per interrupt location). Also, an entry point into a driver is associated with each interrupt.

Each interrupt location corresponds directly to the select code of the device, i.e., if the teleprinter select code is 10_8, the interrupt location in memory is 10_8. The linkage location associated with the device must be higher than the highest select code (interrupt location) used.

Driver Identification Codes

Driver Identification codes are required when creating the EQT and the Interrupt Table. These are the currently defined driver codes:

ØØ to Ø7 Paper Tape Devices:

- ØØ Teleprinter
- Ø1 Tape Reader
- Ø2 Tape Punch

1Ø to 17 Unit Record Devices:

- 1Ø Calcomp Plotter
- 11 Card Reader
- 12 Line Printer
- 15 Mark Sense Card Reader (uses DMA)
- 16 80-Column Line Printer

2Ø to 37 Magnetic Tape/Mass Storage Devices:

21 HP 2Ø2Ø (A or B) Magnetic Tape (7-Track)

22 HP 3Ø3Ø G Magnetic Tape (9-Track) (uses DMA with character packing)

23 HP 797Ø (A or B) Magnetic Tape (9-Track)

4Ø to 77 Instruments

OPERATING INSTRUCTIONS

1. Turn on all desired equipment.

2. Load PREPARE CONTROL SYSTEM (PCS) using the Basic Binary Loader (BBL) or Basic Binary Disc Loader (BBDL).

3. Set starting address 2000_8.

4. Set all switch register bits off; then set switches 5 through 0 to the octal select code (I/O channel) of the teleprinter.

5. Start program execution.

6. Set all switch register bits off.

7. PCS asks for the high-speed input device. (Remember to terminate each reply with a RETURN and LINEFEED.)†

   HS INP?

   Reply with the select code of the high-speed input unit for PCS (either tape reader or teleprinter)

   . . . .

8. PCS asks for high-speed punch

   HS PUN?

   Type the select code of the tape punch or teleprinter to be used by PCS

   . . . .

†*Terminate any reply typed on the keyboard throughout PCS execution with* RETURN LINEFEED. *If an error occurs while typing a response, press* RUBOUT, RETURN LINEFEED, *then retype the response.*

9. PCS asks for the first word of available memory

   FWA MEM?

   Type the octal address beyond the last address necessary for interrupt linkages

   . . . .

10. PCS asks for the last word of available memory

   LWA MEM?

   Type the octal address of last available memory address (first digit must be non-zero)

   . . . .

11. PCS prints

   *LOAD

   At this point, load the appropriate BCS drivers (Magnetic tape first, if present) one at a time. Place the driver tape in the reader and press RUN.

   PCS prints the driver name and absolute memory bounds, then prints *LOAD'and halts for the next tape.

   Keep loading driver tapes until all are loaded. Then load the Input/Output Control routine (IOC), either buffered or non-buffered.

   NOTE: If driver D.21 (HP 2020 (A or B) Magnetic Tape Unit) is loaded, only non-buffered IOC can be used; D.21 turns off the interrupt system. D.11 (Card Reader Driver) and D.23 (HP 7970 (A or B) Magnetic Tape Unit) also require non-buffered IOC when used without DMA.

12. PCS prints IOC and the memory bounds and then asks for Equipment Table entires by printing

   *TABLE ENTRY EQT?

   Press RUN. Then type in the required EQT entries, one per line (each entry followed by RETURN and LINEFEED). Remember that the entries are implicitly assigned octal numbers, starting with 7_8, as they are entered

   xx,D.yy[,D[,U1]]

   NOTE: Elements in brackets "[ ]" are omitted according to the driver requirements.

where

<u>xx</u> = high priority select code of the device

<u>D.yy</u> = driver identification number (see chart).

<u>D</u> = uses DMA; omit if device does not use DMA.

<u>U1</u> = file protect mode for mass storage device; omit if file protect is not desired.

Terminate the EQT by typing

```
/E
```

13. PCS asks for the Standard Unit Table

```
SQT?
```

and requests octal EQT entry numbers (7, 10, 11,...) for the following standard functions:

1. Keyboard input . . . . -KYBD?
2. Teleprinter . . . . -TTY?
3. Library subroutine input at load-time . . . . -LIB?
4. Punch output . . . . -PUNCH?
5. Standard input . . . . -INPUT?
6. Standard list output . . . . -LIST?

Respond to each request by typing the EQT entry number of the device that is most appropriate for the specific function.

14. PCS requests information about the availability of Direct Memory Access

```
DMA?
```

Respond by typing 0 (no DMA), 6 (one channel DMA), or 6, 7 (two channel DMA)

15. PCS halts after typing

```
*LOAD
```

Place the BCS Relocating Loader in the reader and press RUN.

16. PCS loads the Relocating Loader, then prints

```
LOADR
```

and the loader's memory bounds

xxxxx yyyyy

PCS then asks for the Interrupt Linkage Table by printing

INTERRUPT LINKAGE?

and halts.

> Press RUN. Type the Interrupt Linkage Table entries for each device, one per line, in order of ascending select codes.

For a device using only one select code (I/O channel) type

xx,yy,I.zz

where

xx = select code of the device. (Lower numbered of two select codes if device is mass storage.)

yy = octal address of interrupt linkage memory word for the device.

zz = driver identification number (see Table BCS-1).

Example: 10,16,I.00

For a mass storage device using two select codes (I/O channels) type

xx,yy,I.zz
qq,rr,C.zz

where

qq = the lower priority (higher numbered) select code (xx = higher priority, lower numbered select code).

rr = octal address of the interrupt linkage memory word for the device (different from yy).

zz = driver identification number (same as for I.zz).

Example: 11,17,I.21
12,20,C.21

To put an octal instruction (i.e., a precautionary halt instruction) in an unused interrupt location (select code) type

xx,bbbbbb

where

xx = select code

bbbbbb = an octal instruction (b = 0–7).

Example: $15,\underbrace{1020}_{\text{halt instruction}}\overbrace{55}^{\text{halt number}}$

PCS checks each entry after it is typed. If the driver name was typed incorrectly, PCS types

```
*ERROR
```

If the driver was not loaded earlier (step 11) then PCS types

```
*UN name
```

In either of the above cases, refer to Procedure 2 to continue. Terminate the Equipment Table by typing

```
/E
```

17. PCS determines whether there are any undefined references (e.g., to drivers that were not loaded). If none, PCS goes on to the next step.

If some symbols are undefined, PCS prints

```
*UNDEFINED SYMBOLS:
```

followed by a list of entry points for drivers which have been referenced in tables but not loaded

```
I.xx
 .
 .
 .
```

If the drivers were not loaded during step 11 but should have been, restart PCS from step 1. To leave the references unresolved and load in the driver tapes at load-time, (Procedure 3) continue PCS processing with step 18.

NOTE: Drivers that use DMA or entry point IOERR in the loader cannot be left undefined (must be loaded during step 11).

18. PCS lists the entry points of BCS and prints the system linkage area

    *SYSTEM LINK
    xxxxx yyyyy

19. PCS then prints

    *BCS ABSOLUTE OUTPUT

    Check that the tape punch is operable and press RUN. PCS punches a configured BCS tape and halts. To punch additional copies, set switch register bit 15 on and press RUN.

20. Terminate PCS by setting all switch register bits to zero and pressing RUN. PCS halts after printing

    *END

PCS ERROR HALTS AND MESSAGES H

Halt Code	Meaning	Action
1Ø2Ø55	A line is about to be printed on the teleprinter	Turn punch unit OFF. Press RUN.
1Ø2Ø56	A line has been printed while the teleprinter punch unit was off.	Turn punch unit ON. Press RUN.
1Ø2Ø66	Tape supply low on tape punch which is producing absolute binary output. Trailer follows last valid output.	Place new reel of tape in unit. Press RUN. Leader is punched.
1Ø2Ø77	BCA tape is punched	To produce additional copies, set switch 15 on.

Message	Meaning	Action
*EOT	End-of-tape	Place next tape in tape reader and press RUN to continue loading.
*ERROR	A non-numeric parameter or illegal numeric parameter has been entered.	Retype the entire entry correctly.
I/O DRIVER? D.rr	A driver has been named in EQT entry but has not been loaded.	1. If the driver is to be loaded with user's program at load-time, type an exclamation mark (!). The driver name is added to the loader's LST. 2. If the driver should have been loaded, restart PCS.

Message	Meaning	Action
*LØ1	Checksum error	To reread record, reposition tape to beginning of record and press RUN. If computer halts again, tape must be replaced.
*LØ2	Illegal record read: The last record read was not recognized as a valid relocatable format record.	To reread record, reposition tape to beginning of record and press RUN. If computer halts again, tape must be replaced.
*LØ3	Memory overflow: The length of BCS exceeds available memory'	Abort PCS. Reduce the number of core resident I/O drivers or increase memory.
*LØ4	System linkage area overflow in Base Page.	Abort PCS. Reduce the number of, or reorder the core resident I/O drivers.
*LØ5	Loader symbol table overflow: The number of EXT/ENT symbols exceeds available memory.	Abort PCS. Reduce the number of, or reorder the core resident I/O drivers.
*LØ6	PCS interprets the program length of BCS to be zero.	Abort PCS.
*LØ7	Duplicate entry points; an entry point in the current program matches a previously loaded entry point.	Eliminate an entry point. Check to see if the same program was loaded twice.
*UNDEFINED SYMBOL: symbol	An entry point in a BCS module cannot be located.	If the subroutine should have been loaded, rerun PCS.
*UN name	The name I.ee is not defined as an entry point in any I/O driver previously loaded.	1. If the driver name was typed incorrectly, retype the entire entry correctly. 2. If the driver is to be loaded with the user's program at load-time, type an exclamation mark (!).

HOW TO USE BCS TO RELOCATE AND RUN PROGRAMS

BCS performs two main functions: 1) relocates and links subroutines to main programs, and 2) executes programs.

Starting with relocatable code produced by an assembler or compiler, there are two possible methods to accomplish function 1 and reach function 2:

a. BCS relocates the code (including subroutines) into core memory directly and then executes it.

b. BCS relocates the code (including subroutines) and punches it onto an absolute tape along with the necessary system routines, drivers, tables, etc. This absolute tape can then be loaded into core through BBL or BBDL and executed.

Method (a) is faster, but does not provide a permanent, runnable copy of the program. Not only does the program code have to be relocated each time the program is to be run, but less core is available because the Relocating Loader occupies a part of memory.

Method (b) takes longer the first time, but provides a permanent copy of the program that can be executed. Also, more core is available since the program can (at run-time) use the space occupied by the Relocating Loader at load-time.

OPERATING INSTRUCTIONS

1. Load a configured BCS into core with BBL or BBDL. (See Procedure 1 for generation of a configured BCS.)

2. Set a starting address of 2_8.

3. Set all switch register bits off, then select the following options:

   Bit 15 on (suppress memory allocation listing)
   off (include memory allocation listing)

   Bit 14 on (punch absolute tape copy of program)
   off (relocate into core, do not punch tape)

If Bit 14 on and a teleprinter is to be used for punching, then

Bit 13 on (teleprinter is a 2754B and can print and punch separately; set teleprinter mode to KT)

off (teleprinter cannot print and punch separately; BCS halts before and after each line of printing so that the operator can turn on/off punch unit to avoid punching list output, then punch the absolute binary output).

4. Place the first relocatable program tape into the reader. Press PRESET and RUN. BCS reads and relocates the binary code on the tape. If switch register bit 14 is on, an absolute binary tape is punched. (Otherwise, BCS relocates the program in memory.)

5. BCS halts after typing

   *LOAD

   Load the user relocatable tapes as follows: Set switch register bits 2 - 0 off.

   Place the tape in the reader. Set switch register bit 15 on (if desired) to suppress memory allocation listing. Press RUN. When tape has been read, BCS halts after printing

   *LOAD

   If there are more user tapes to load, repeat step 5.

6. After all user program tapes have been loaded, there are several options:

   To read a library subroutine tape (and load only those subroutines which are necessary to resolve externals). (Step 7)

   To list undefined externals (or bypass further loading if there are no undefined externals). (Step 8)

   To bypass further loading even if undefined externals remain. (Step 9)

7. Set switch register bit 2 on (bits 1 and 0 off). Place the relocatable library tape in the reader (FORTRAN IV library must be loaded first). Set switch register bit 15 on to suppress the memory allocation listing, if desired. Press RUN.

When the tape has been read, BCS halts after indicating:

```
No undefined externals
     *LST
```

(Set switch register bit 2 off and go to step 10.)

or

```
Undefined externals
     symbol
     symbol
       .
       .
       .
     symbol
     *LOAD
```

Return to Step 6 and select an option.

8. Set switch register bit 0 on (bits 1 and 2 off). Press RUN. BCS indicates whether undefined externals exist by printing either:

```
No undefined externals
     *LST
```

(Set switch register bit 2 off and go to Step 10)

or

```
Undefined externals
     symbol
     symbol
       .
       .
     symbol
     *LOAD
```

Return to Step 6.

9. Set switch register bit 1 on (bits 2 and 0 off). Press RUN. BCS goes on to Step 10, even though undefined externals may still exist.

10. BCS has completed loading and is ready to print the Loader Symbol Table (LST), common bounds, and linkage area bounds. Set switch register bit 15 on to suppress listing of these items. Set bit 15 off to list them.

    If a 2754B Teleprinter is used, set the mode switch to "T" to enable the tape punch.

    Press RUN.

11. BCS completes listing (if requested by bit 15).

    If the program was relocated into core (bit 14 off), BCS prints

    ```
    *RUN
    ```

    Press RUN to execute the program.

12. If the program was punched onto paper tape (bit 14 on), BCS prints

    ```
    *END
    ```

13. Tear off the absolute tape output and wind. To execute the program:

    Load the tape with BBL or BBDL.

    Start the program at location 2_8.

BCS ERROR HALTS AND MESSAGES J

LOAD-TIME

Error Halt	Meaning	Action
1Ø2Ø55	A line is about to be printed on the teleprinter.	Turn punch unit OFF. Press RUN.
1Ø2Ø56	A line has just been printed on the teleprinter with the tape punch OFF.	Turn punch unit ON. Press RUN.
1Ø2Ø66	Tape supply low on tape punch which is producing absolute binary output. Trailer follows last valid output	Place new reel of tape in unit. Press RUN. Leader is punched.

Message	Meaning	Action
*LØ1	Checksum error.	To reread record, reposition tape to beginning of record and press RUN. If computer halts again, tape must be replaced.
*LØ2	Illegal record read: The last record read was not recognized as a valid relocatable record tape.	To reread record, reposition tape to beginning of record and press RUN. If computer halts again, tape must be replaced.
*LØ3	Memory overflow: The length of BCS exceeds available memory.	Abort load. Reduce program size or increase memory
*LØ4	System linkage area overflow in Base Page.	Abort load. Reduce program size or alter subprogram loading sequence.

Message	Meaning	Action
*LØ5	Loader symbol table overflow: The number of EXT/ ENT symbols exceeds available memory.	Abort load. Reduce program size or increase memory.
*LØ6	Common block error: The length of the common block in the current program is greater than the length of the first common block allocated	Abort load. Reorder the programs during loading or make the common blocks the same length.
*LØ 7	Duplicate entry points: An entry point in the current program matches a previously declared entry point.	Abort load. Eliminate an entry point. Check to see if the same program was loaded twice.
*LØ8	No transfer address: The initial starting location was not present in any of the programs which were loaded.	Load the absolute starting address into the A-register. Start program execution.
*LØ9	Record out of sequence: A NAM record was encountered before the previous program was terminated with an END record.	1. Reload the program. 2. If program does not load properly, replace the binary tape for the program being loaded.

RUN-TIME

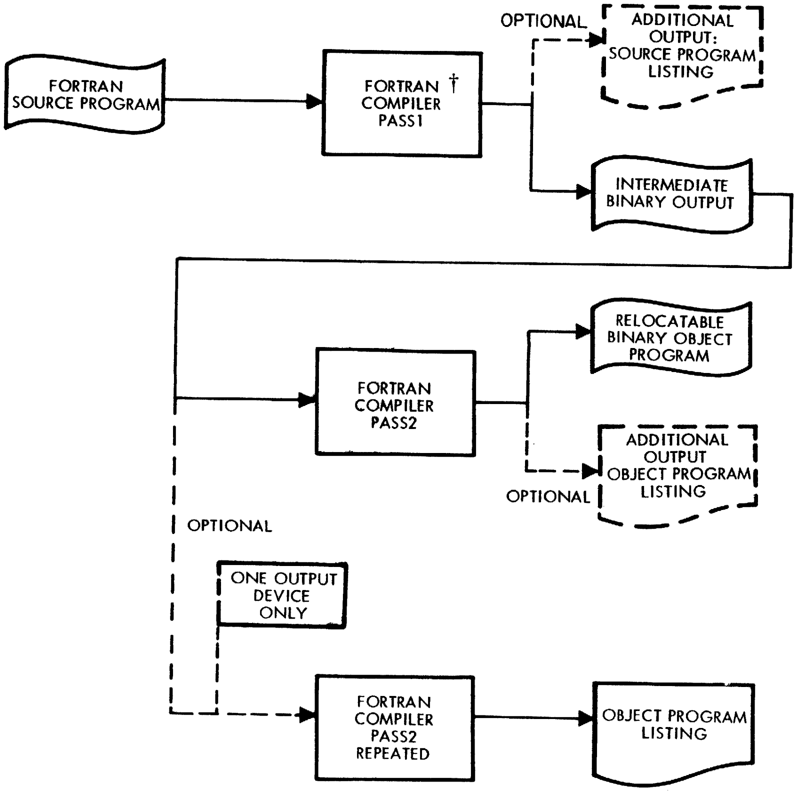
Certain library routines, including the Formatter, produce error messages at run-time.

Halt Code	Meaning
1Ø6Ø55	Program has attempted to execute a non-program area of core. Warning-only. Program can be restarted.

In the HP 9625C Real-Time Executive System, the 2100A mobilizes disc storage, data acquisition subsystems, instruments and computer peripherals into a powerful real-time multiprogramming system.

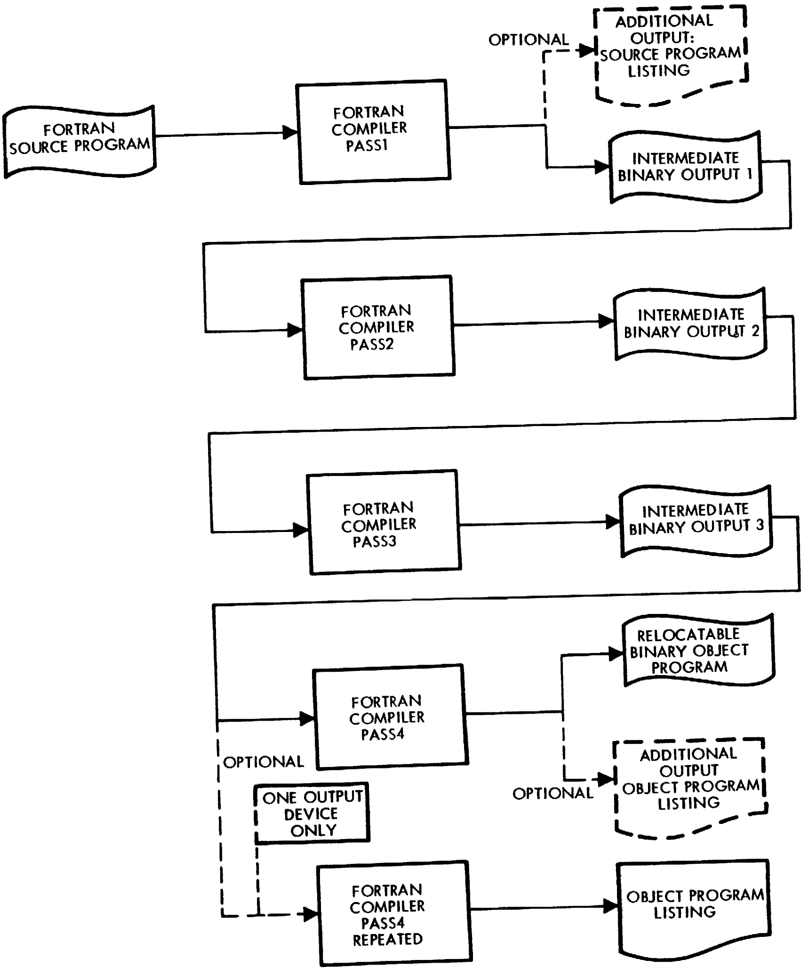
FORTRAN Reference Manual

CONTENTS


FORTRAN SOURCE PROGRAM
FORTRAN † COMPILER PASS1
OPTIONAL
ADDITIONAL OUTPUT: SOURCE PROGRAM LISTING
INTERMEDIATE BINARY OUTPUT
RELOCATABLE BINARY OBJECT PROGRAM
FORTRAN COMPILER PASS2
ADDITIONAL OUTPUT OBJECT PROGRAM LISTING
OPTIONAL
OPTIONAL
ONE OUTPUT DEVICE ONLY
FORTRAN COMPILER PASS2 REPEATED
OBJECT PROGRAM LISTING

†When compiling with the magnetic tape system, operator intervention ceases after Pass 1 has been loaded.

8K MEMORY
FORTRAN COMPILATION PROCESS


OPTIONAL
ADDITIONAL OUTPUT: SOURCE PROGRAM LISTING
FORTRAN SOURCE PROGRAM
FORTRAN COMPILER PASS1
INTERMEDIATE BINARY OUTPUT 1
FORTRAN COMPILER PASS2
INTERMEDIATE BINARY OUTPUT 2
FORTRAN COMPILER PASS3
INTERMEDIATE BINARY OUTPUT 3
RELOCATABLE BINARY OBJECT PROGRAM
FORTRAN COMPILER PASS4
OPTIONAL
ADDITIONAL OUTPUT OBJECT PROGRAM LISTING
OPTIONAL
ONE OUTPUT DEVICE ONLY
FORTRAN COMPILER PASS4 REPEATED
OBJECT PROGRAM LISTING

4K MEMORY
FORTRAN COMPILATION PROCESS

INTRODUCTION

The FORTRAN compiler system accepts as input, a source program written according to American Standard Basic FORTRAN specifications; it produces as output, a relocatable binary object program which can be loaded and executed under control of an HP operating system.

In addition to the ASA Basic FORTRAN language, HP FORTRAN provides a number of features which expand the flexibility of the system. Included are:

> Free Field Input: Special characters included with ASCII input data direct its formatting; a FORMAT statement need not be specified in the source program.
>
> Specification of heading and editing information in the FORMAT statement through use of the "..." notation; permits alphanumeric data to be read or written without giving the character count.
>
> Array declaration within a COMMON statement.
>
> Redefinition of its arguments and common areas by a function subprogram.
>
> Interpretation of an END statement as a RETURN statement.
>
> Basic External Functions which perform masking (Boolean) operations.
>
> Two-branch IF statement.
>
> Octal constants.

There are several versions of the HP FORTRAN Compiler; each is designed to run in a different operating environment: Software Input/Output System, etc. The operating system manuals contain descriptions of any features limited to special versions of the compiler.

PROGRAM FORM 1

A FORTRAN program is constructed of characters grouped into lines and statements.

1.1 CHARACTER SET

The program is written using the following characters:

Alphabetic:	A through Z
Numeric:	0 through 9
Special:	
	Space
=	Equals
+	Plus
-	Minus
*	Asterisk
/	Slash
(	Left Parenthesis
)	Right Parenthesis
,	Comma
.	Decimal Point
$	Dollar Sign
"	Quotation mark

Spaces may be used anywhere in the program to improve appearance; they are significant only within heading data of FORMAT statements and, in lieu of other information, in the first six positions of a line.

In addition to the above set which is used to construct source language statements, certain characters have special significance when appearing with ASCII input data. They are the following:

space,	Data item delimiters
/	Record terminator
+ -	Sign of item
. E + -	Floating point number
@	Octal integer
". . ."	Comments
←	Suppresses CR and LF (output)

Details on the input data character set are given in Chapter 7.

1.2 LINES

A line is a sequence of up to 72 characters. On paper tape, each line is terminated by a *return*, CR , followed by a *line feed*, LF . This terminator may be in any position following the statement information or comment contained in the line. If an error is punched on a paper tape, a *rubout* before the *return* and *line feed* causes the entire line containing the error to be ignored.

Statements

A statement may be written in an initial line and up to five continuation lines. The statement may occupy positions 7 through 72 of these lines. The initial line contains a zero or blank in position 6. A continuation line contains any character other than zero or space in position 6 and may not contain a C in position 1.

Statement Labels

A statement may be labeled so that it may be referred to in other statements. A label consists of one to four numeric digits placed in any of the first five positions of a line. The number is unsigned and in the range of 1 through 9999. Imbedded spaces and leading zeros are ignored. If no label is used, the first five positions of the statement line must be blank. The statement label or blank follows the CR LF terminator of the previous line.

Comments

Lines containing comments may be included with the statement lines; the comments are printed along with the source program listing. A comment line requires a C in position 1 and may occupy positions 2 through 72. If more than one line is used, each line requires a C indicator. Each comment line is terminated with a CR and LF .

Control Statement

The first statement of a program is the control statement; it defines the output to be produced by the FORTRAN compiler. The following options are available:

Relocatable binary — The program can be loaded by the relocating loader and run.

Source Listing output — A listing of the source program is produced.

Object Listing output — A list of the object program is produced.

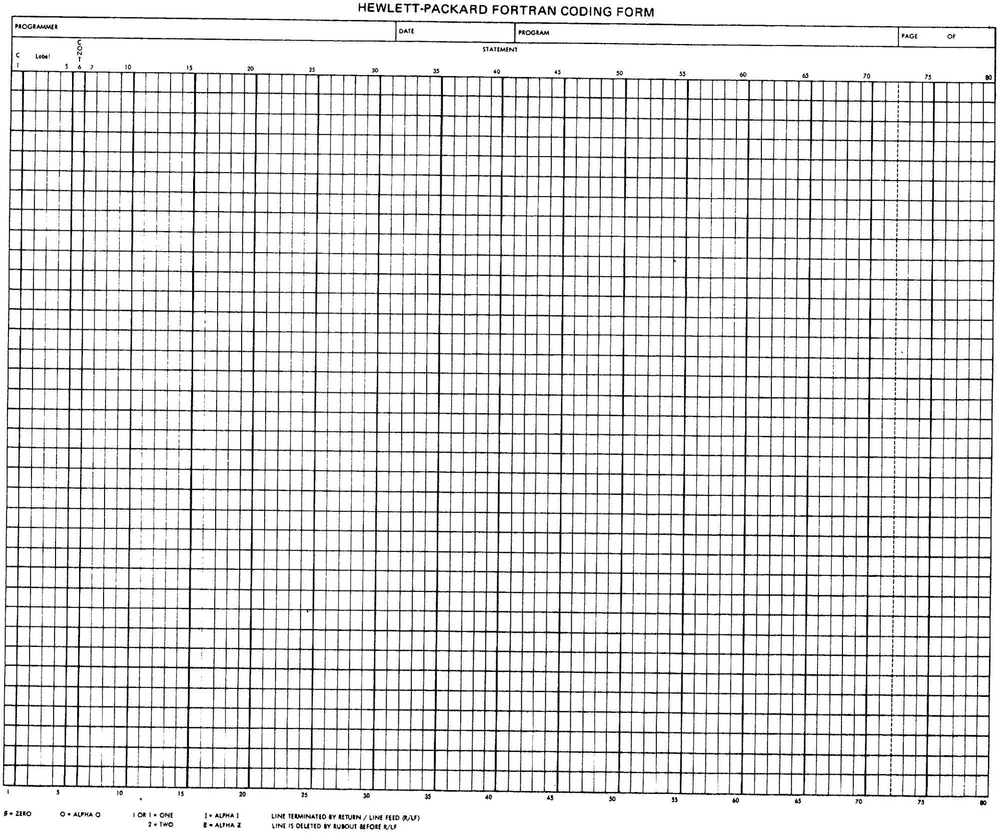
The control statement must be followed by the CR LF terminator.

End Line

Each subprogram is terminated with an end line which consists of blanks in positions 1 through 6 and the letters E, N, and D located in any of the positions 7 through 72. The special end line, END$, signifies the end of five or less programs being compiled at one time. The end line is terminated by CR LF .

1.3 CODING FORM

The FORTRAN coding form is shown below. Columns 73-80 may be used to indicate a sequence number for a line; they must not be punched on paper tape. All other columns of the form conform with line positions for paper tape.



HEWLETT-PACKARD FORTRAN CODING FORM

PROGRAMMER

DATE

PROGRAM

PAGE OF

C Label CONT STATEMENT

1 5 6 7 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80

M0251

Ø = ZERO

O = ALPHA O

I OR I = ONE

2 = TWO

I = ALPHA I

Z = ALPHA Z

LINE TERMINATED BY RETURN / LINE FEED (R/LF)

LINE IS DELETED BY RUBOUT BEFORE R/LF

SAMPLE CODING FORM
(Actual Size 11 x 13-1/2)

ELEMENTS OF HP FORTRAN 2

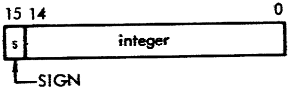
HP FORTRAN processes two types of data. They differ in mathematical significance, constant format, and symbolic representation. The two types are real and integer quantities.

2.1 DATA TYPE PROPERTIES

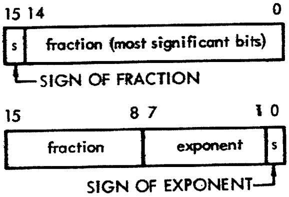
Integer and real data quantities have different ranges of values.

An integer quantity has an assumed fixed decimal point. It is represented by a 16-bit computer word with the most significant bit as the sign and the assumed decimal point on the right of the least significant bit.

An integer quantity has a range of -2^{15} to 2^{15} -1.



A real quantity has a floating decimal point; it consists of a fractional part and an exponent part. It is represented by two 16-bit computer words; the exponent and its sign are eight bits; the fraction and its sign are twenty-four bits.



It has a range in magnitude of approximately 10^{-38} to 10^{38} and may assume positive, negative, or zero values. If the fraction is negative, the number is in two's complement form. A zero value is stored as all zero bits. Precision is approximately seven decimal digits.

2.2 CONSTANTS

A constant is a value that is always defined during execution and may not be redefined. Three types of constants are used in HP FORTRAN: integer, octal (treated as integer), and real. The type of constant is determined by its form and content.

Integer

An integer constant consists of a string of up to five decimal digits. If the range -32768 to 32767 (-2^{15} to 2^{15} -1) is exceeded, a diagnostic is provided by the compiler.

Examples:

8364
1720
1872
125
3653

5932
9
31254
1
30000

Octal

Octal constants consist of up to six octal digits followed by the letter B. The form is:

$n_1\ n_2\ n_3\ n_4\ n_5\ n_6$ B

n_1 is 0 or 1

$n_2 - n_6$ are 0 through 7

If the constant exceeds six digits, or if a non-octal digit appears, the constant is treated as zero and a compiler diagnostic is provided.

Examples:

7677B
3270B
3520B
175B
567B

7631B
5B
75026B
177776B
177777B

Real

Real constants may be expressed as an integer part, a decimal point, and a decimal fraction part. The constant may include an exponent, representing a power of ten, to be applied to the preceding quantity. The forms of real constants are:

```
n.n   n.   .n   n.nE±e   n.E±e   .nE±e
```

n is the number and e is the exponent to the base ten. The plus sign may be omitted for a positive exponent. The range of e is 0 through 38. When the exponent indicator E is followed by a + or - sign, then all digits between the sign and the next operator or delimiter are assumed to be part of the exponent expression, e.

If the range of the real constant is exceeded, the constant is treated as zero and a compiler diagnostic message occurs.

Examples:

```
4.512      4.5E2
4.         .45E+3
.512       4.5E-5
4.0        0.5
4.E-10     .5E+37
1.         10000.0
```

2.3 VARIABLES

A variable is a quantity that may change during execution; it is identified by a symbolic name. Simple and subscripted variables are recognized. A simple variable represents a single quantity; a subscripted variable represents a single quantity (element) within an array of quantities. Variables are identified by one to five alphanumeric characters; the first character must be alphabetic.

The type of variable is determined by the first letter of the name. The letters I, J, K, L, M, and N, indicate an integer (fixed point) variable; any other letter indicates a real (floating point) variable. Spaces imbedded in variable names are ignored.

Simple Variable

A simple variable defines the location in which values can be stored. The value specified by the name is always the current value stored in that location.

Examples:

Integer	Real
I	ALPHA
JAIME	G13
K9	DOG
MIL	XP2
NIT	GAMMA

Subscripted Variable

A subscripted variable defines an element of an array; it consists of an alphanumeric identifier with one or two associated subscripts enclosed in parentheses. The identifier names the array; the subscripts point to the particular element. If more than two subscripts appear, a compiler diagnostic message is given.

Subscripts may be integer constants, variables, or expressions; they may have the form (exp_1, exp_2), where exp_i is one of the following:

c*v+k
c*v-k
c*v
v+k
v-k
v
k

where c and k are integer constants and v is a simple integer variable.

Examples:

Integer	Real
I(J, K)	A(J)
LAD(3, 3)	BACK(M+5, 9)
MAJOR (24*K, I+5)	OP45(4*I)
NU (K+2)	RADI (IDEG)
NEXT (N*5)	VOLTI (I, J)

2.4 ARRAYS

An array is an ordered set of data of one or two dimensions; it occupies a block of successive memory locations. It is identified by a symbolic name which may be used to refer to the entire array. An array and its dimensions must be declared at the beginning of the program in a DIMENSION or COMMON statement. The type of an array is determined by the first letter of the array name. The letters I, J, K, L, M, and N, indicate an integer array; any other letter indicates a real array.

Each element of an array may be referred to by the array name and the subscript notation. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array, however, no diagnostic messages are issued.

Array Structure

Elements of arrays are stored by columns in ascending order of storage locations. An array declared as SAM(3, 3), would be structured as:

		Columns	
	SAM(1, 1)	SAM(1, 2)	SAM(1, 3)
Rows	SAM(2, 1)	SAM(2, 2)	SAM(2, 3)
	SAM(3, 1)	SAM(3, 2)	SAM(3, 3)

and would be stored as:

m	SAM(1, 1)
m+1	SAM(2, 1)
m+2	SAM(3, 1)
m+3	SAM(1, 2)
m+4	SAM(2, 2)
m+5	SAM(3, 2)
m+6	SAM(1, 3)
m+7	SAM(2, 3)
m+8	SAM(3, 3)

The location of an array element with respect to the first element is a function of the subscripts, the first dimension, and the type of the array. Addresses are computed modulo 2^{15} .

Given DIMENSION A (L, M), the memory location of A (i, j) with respect to the first element, A, of the array, is given by the equation:

$$\ell = A + [\ i - 1 + L(j - 1)\ ] *S$$

The quantity in brackets is the expanded subscript expression. The element size, s , is the number of storage words required for each element of the array: for integer arrays, s = 1 ; for real arrays, s = 2 .

Array Notation

The following subscript notations are permitted for array elements:

For a two-dimensional array, $A(d_1 , d_2)$:

A(I, J)	implies A(I, J)
A(I)	implies A(I, 1)
A	implies A(1, 1)†

For a single-dimension array, A(d)

A(I)	implies A(I)
A	implies A(1)

The elements of a single-dimension array, A(d), however, may not be referred to as A(I, J). A diagnostic message is given by the compiler if this is attempted.

2.5 EXPRESSIONS

An expression is a constant, variable, function or a combination of these separated by operators and parentheses, written to comply with the rules for constructing the particular type of instruction. An arithmetic expression has numerical value; its type is determined by the type of the operands.

† In an Input/Output list, the name of a dimensioned array implies the entire array rather than the first element.

Examples:

```
A+B-C              .4+SIN(ALPHA)
X*COS(Y)           A/B+C-D*F
RALPH-ALPH         4+2*IABS(LITE)
```

2.6 STATEMENTS

Statements are the basic functional units of the language. Executable statements specify actions; non-executable statements describe the characteristics and arrangement of data, editing information, statement functions, and classification of program units.

A statement may be given a numeric label of up to four digits (1 to 9999); a label allows other statements to refer to a statement. Each statement label used must be unique within the program.

ARITHMETIC EXPRESSIONS AND ASSIGNMENT STATEMENTS 3

3.1 ARITHMETIC EXPRESSIONS

An arithmetic expression may be a constant, a simple or subscripted variable, or a function. Arithmetic expressions may be combined by arithmetic operators to form complex expressions.

Arithmetic operators are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

If α is an expression, (α) is an expression.
If α and β are arithmetic expressions, then the following are expressions:

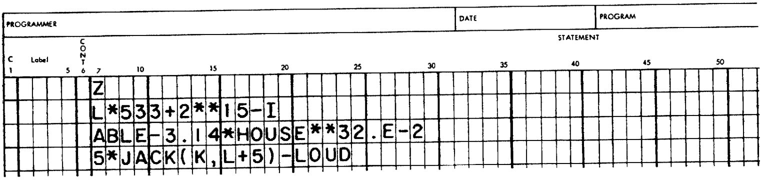
$$\alpha + \beta \qquad \alpha - \beta \qquad \alpha/\beta$$
$$\alpha * \beta \qquad +\alpha \qquad -\alpha$$
$$\alpha ** \beta$$

An arithmetic expression may not contain adjoining arithmetic operators, α op op β.

Expressions of the form $\alpha**\beta$ and $\alpha**(-\beta)$ are valid; $\alpha**\beta**\gamma$ is not valid.

Integer overflow resulting from arithmetic operations is not detected at execution time.

Examples:


PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1
5
6
7
10
15
20
25
30
35
40
45
50

```
Z
L*533+2**15-I
ABLE-3.14*HOUSE**32.E-2
5*JACK(K,L+5)-LOUD
```

Order or Evaluation

In general, the hierachy of arithmetic operation is:

**	exponentiation	class 1
/	division	class 2
*	multiplication	
-	subtraction	class 3
+	addition	

In an expression with no parentheses or within a pair of parentheses, evaluation basically proceeds from left to right, or in the above order if adjacent operators are in a different class. †

Expressions enclosed in parentheses and function references are evaluated as they are encountered from left to right.

Examples:

In the examples below, $s_1, s_2, \ldots, s_n$ indicate intermediate results during the evaluation of the expression; the symbol → can be interpreted as "goes to".

a) Evaluation of class 1 precedes class 3

A+B**C-D

$B**C \rightarrow s_1$

$s_1 + A \rightarrow s_2$

$s_2 - D \rightarrow s_3$ s_3 is the evaluated expression

b) Evaluation of class 2 precedes class 3

A*B*C/D+E*F-G/H

$A*B \rightarrow s_1$

$s_1 * C \rightarrow s_2$

$s_2 / D \rightarrow s_3$

$E*F \rightarrow s_4$

$s_4 + s_3 \rightarrow s_5$

$G/H \rightarrow s_6$

$-s_6 \rightarrow s_7$

$s_7 + s_5 \rightarrow s_8$ s_8 is the evaluated expression

† When writing an integer expression it is important to remember not only the left to right scanning process, but also that dividing an integer quantity by an integer quantity yields a truncated result; thus 11/3 = 3 . The expression I*J/K may yield a different result than the expression J/K*I . For example, 4*3/2 = 6 ; but 3/2*4 = 4 .

c) Evaluation of an expression including a function is performed.

A+B**C+D+COS(E)
$B**C \rightarrow s_1$
$A+s_1 \rightarrow s_2$
$s_2 + D \rightarrow s_3$
$COS(E) \rightarrow s_4$
$s_4 + s_3 \rightarrow s_5$ s_5 is the evaluated expression

d) Parentheses can control the order of evaluation

A*B/C+D
$A*B \rightarrow s_1$
$s_1 /C \rightarrow s_2$
$s_2 +D \rightarrow s_3$ s_3 is the evaluated expression

A*B/(C+D)
$A*B \rightarrow s_1$
$C+D \rightarrow s_2$
$s_1 /s_2 \rightarrow s_3$ s_3 is the evaluated expression

e) If more than one pair of parentheses or if an exponential expression appears, evaluation is performed left to right.

A+B**C-(D*E+F)+(G-H*P)
$B**C \rightarrow s_1$
$s_1 + A \rightarrow s_2$
$D*E \rightarrow s_3$
$s_3 +F \rightarrow s_4$
$-s_4 \rightarrow s_5$
$s_5 + s_2 \rightarrow s_6$
$H*P \rightarrow s_7$
$-s_7 \rightarrow s_8$
$s_8 + G \rightarrow s_9$
$s_9 + s_6 \rightarrow s_{10}$ s_{10} is the evaluated expression

Type of Expression

With the exception of exponentiation and function arguments, all operands within an expression must be of the same type. An expression is either real or integer depending on the type of all of its constituent elements.

If either an integer or real operand is exponentiated by an integer operand, the resultant element is of the same type as that of the operand being exponentiated. If both operands are real, the resultant element is real.

Examples:

J**I	integer
A**I	real
A**B	real

An integer exponentiated by a real operand is not valid.

3.2 ASSIGNMENT STATEMENTS

An arithmetic assignment statement is of the form:

```
v = e
```

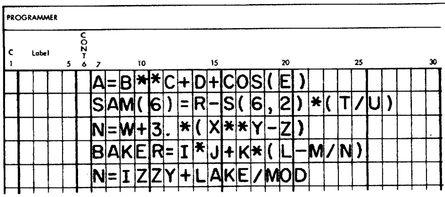
The variable, v , may be simple or subscripted; e is an expression. Execution of this statement causes the evaluation of the expression, e , and the assignment of the value to the variable.

Type of Statement

The processing of the evaluated expression is performed according to the following table:

Type of v	Type of e	Assignment rule
Integer	Integer	Transmit e to v without change.
Integer	Real	Truncate and transfer as integer to v.
Real	Integer	Transform integer form of e to floating decimal and transfer to v.
Real	Real	Transmit e to v without change.

Examples:


PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20
25
30

```
      A=B**C+D+COS(E)
      SAM(6)=R-S(6,2)*(T/U)
      N=W+3.*(X**Y-Z)
      BAKER=I*J+K*(L-M/N)
      N=IZZY+LAKE/MOD
```

Transmit without change
Transmit without change
Truncate
Convert to real
Transmit without change

3.3 MASKING OPERATIONS

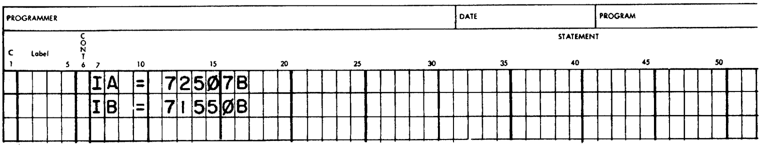
In HP FORTRAN, masking operations may be performed using the Basic External Functions IAND, IOR, and NOT (see Chapter 6). These functions are as follows:

IAND	Form the bit-by-bit logical product of two operands
IOR	Form the bit-by-bit logical sum of two operands
NOT	Complement the operand

The operations are described by the following table:

Value of Arguments		Value of Function		
a_1	a_2	IAND (a_1, a_2)	IOR (a_1, a_2)	NOT (a_1)
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Examples:


PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1
5
6
7
10
15
20
25
30
35
40
45
50

```
      IA = 725Ø7B
      IB = 7155ØB
```

IAND (IA, IB) is 70500B
IOR (IA, IB) is 73557B
NOT (IA) is 105270B

SPECIFICATIONS STATEMENTS 4

The Specifications statements, which include DIMENSION, COMMON, and EQUIVALENCE, define characteristics and arrangement of the data to be processed. These statements are non-executable; they do not produce machine instructions in the object program. The statements must all appear before the first executable statement in the following order: DIMENSION, COMMON, and EQUIVALENCE.

4.1 DIMENSION

The DIMENSION statement reserves storage for one or more arrays.

DIMENSION v_1 (i_1), v_2 $(i_2), \ldots,$ v_n (i_n)

An array declarator, $v_j(i_j)$; defines the name of an array, v_j , and its associated dimensions, (i_j) . The declarator subscript, i , may be an integer constant or two integer constants separated by a comma. The magnitude of the values given for the subscripts indicates the maximum value that the subscript may attain in any reference to the array.

The number of computer words reserved for a given array is determined by the product of the subscripts and the type of the array name. For integer arrays, the number of words equals the number of elements in the array. For real arrays, two words are used for each element; the storage area is twice the product of the subscripts.

A diagnostic message is printed if an array size exceeds 2^{15} -1 locations.

Examples:

DIMENSION SAM (5, 10), ROGER (10, 10), NILE (5,20)

Area reserved for SAM	5*10*2	= 100 words
Area reserved for ROGER	10*10*2	= 200 words
Area reserved for NILE	5*20*1	= 100 words

4.2 COMMON

The COMMON statement reserves a block of storage that can be referenced by the main program and one or more subprograms. The areas of common information are specified by the statement form:

$$\text{COMMON } a_1, a_2, \ldots, a_n$$

Each area element, a_i , identifies a segment of the block for the subprogram in which the COMMON statement appears. The area elements may be simple variable identifiers, array names, or array declarators (dimensioned array names).

If dimensions for an array appear both in a COMMON statement and a DIMENSION statement, those in the DIMENSION statement will be used.

Any number of COMMON statements may appear in a subprogram section (preceding the first executable statement). The order of the arrays in common storage is determined by the order of the COMMON statements and the order of the area elements within the statements. All elements are stored contiguously in one block.

At the beginning of program execution, the contents of the common block are undefined; the data may be stored in the block by input/output or assignment statements.

Examples:

COMMON I (5), A (6), B (4)

Area reserved for I	=	5 words
Area reserved for A	=	12 words
Area reserved for B	=	8 words
Common area		25 words

	Common Block
Origin	I (1)
	I (2)
	I (3)
	I (4)
	I (5)
	A (1)
	A (1)

A (2)
A (2)
A (3)
A (3)
A (4)
A (4)
A (5)
A (5)
A (6)
A (6)
B (1)
B (1)
B (2)
B (2)
B (3)
B (3)
B (4)
B (4)

Correspondence of Common Blocks

Each subprogram that uses the common block must include a COMMON statement. Each subprogram may assign different variable and array names, and different array dimensions, however, if corresponding quantities are to agree, the types should be the same for corresponding positions in the block.

Examples:

```
MAIN PROG COMMON I (5), A (6), B (4)
     .
     .
     .
SUBPROG1 COMMON J (3), K (2), C (5), D (5)
```

MAIN PROG reference	Common Block	SUBPROG1 reference
I (1)	integer 1	J (1)
I (2)	integer 2	J (2)
I (3)	integer 3	J (3)
I (4)	integer 4	K (1)
I (5)	integer 5	K (2)
A (1)	real 1	C (1)
A (1)	real 1	C (1)

MAIN PROG reference	Common Block	SUBPROG1 reference
A (2)	real 2	C (2)
A (2)	real 2	C (2)
A (3)	real 3	C (3)
A (3)	real 3	C (3)
A (4)	real 4	C (4)
A (4)	real 4	C (4)
A (5)	real 5	C (5)
A (5)	real 5	C (5)
A (6)	real 6	D (1)
A (6)	real 6	D (1)
B (1)	real 7	D (2)
B (1)	real 7	D (2)
B (2)	real 8	D (3)
B (2)	real 8	D (3)
B (3)	real 9	D (4)
B (3)	real 9	D (4)
B (4)	real 10	D (5)
B (4)	real 10	D (5)

If portions of a common block are not referred to by a particular subprogram, dummy variables may be used to provide correspondence in reserved areas.

Examples:

```
MAIN PROG COMMON I (5), A (6), B (4)
          .
          .
          .
SUBPROG2 COMMON J (17), B (4)
```

MAIN PROG reference	Common Block	SUBPROG2 reference
I (1)	integer 1	J (1)
I (2)	integer 2	J (2)
I (3)	integer 3	J (3)
I (4)	integer 4	J (4)
I (5)	integer 5	J (5)

MAIN PROG reference	Common block	SUBPROG2 reference
A (1)	real 1	J (6)
A (1)	real 1	J (7)
A (2)	real 2	J (8)
A (2)	real 2	J (9)
A (3)	real 3	J (10)
A (3)	real 3	J (11)
A (4)	real 4	J (12)
A (4)	real 4	J (13)
A (5)	real 5	J (14)
A (5)	real 5	J (15)
A (6)	real 6	J (16)
A (6)	real 6	J (17)
B (1)	real 7	B (1)
B (1)	real 7	B (1)
B (2)	real 8	B (2)
B (2)	real 8	B (2)
B (3)	real 9	B (3)
B (3)	real 9	B (3)
B (4)	real 10	B (4)
B (4)	real 10	B (4)

J (17) is a dummy array. It is not referenced in SUBPROG 2 but provides proper correspondence in reserved areas so that SUBPROG 2 can refer to array B.

The length of the common block may differ in different subprograms, however, the subprogram (or main program) with the longest common block must be the first to be loaded at execution time.

4.3 EQUIVALENCE

The EQUIVALENCE statement permits sharing of storage by two or more entities. The statement has the form:

EQUIVALENCE (k_1), $(k_2), \ldots, (k_n)$

in which each k is a list of the form:

$a_1, a_2, \ldots, a_m$

Each a is either a variable name or a subscripted variable; the subscript of which contains only constants. The number of subscripts must correspond to the number of subscripts for the related array declarator.

All names in the list may be used to represent the same location. If an equivalence is established between elements of two or more arrays, there is a corresponding equivalence between other elements of the arrays; the arrays share some storage locations. The lengths may be different or equal.

Examples:

```
DIMENSION A (5), B (4)

EQUIVALENCE (A (4), B (2))
```

Array 1 Name	Array 2 Name	Quantity Element
A (1)		real 1
		real 1
A (2)		real 2
		real 2
A (3)	B (1)	real 3
		real 3
A (4)	B (2)	real 4
		real 4
A (5)	B (3)	real 5
		real 5
	B (4)	real 6
		real 6

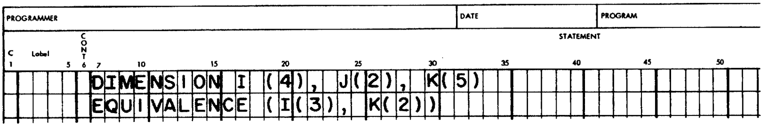
The EQUIVALENCE statement establishes that the names A (4) and B (2) identify the fourth real quantity. The statements also establish a similar correspondence between A (3) and B (1), and A (5) and B (3).

An integer array or variable may be made equivalent to a real array or variable; equivalence may be established between different types. The variables may be with or without subscripts.

The effect of an EQUIVALENCE statement depends on whether or not the variables are assigned to the common block. When two variables or array elements share storage, the symbolic names of the variables or arrays may not both appear in COMMON statements in the same subprogram. The assignment of storage to variables and arrays declared in a COMMON statement is determined on the basis of their type and the array declarator. Entities so declared are always contiguous according to the order in the COMMON statement. The EQUIVALENCE statement must not alter the origin of the common block, but arrays may be defined so that the length of the common block is increased.

Examples:

a) Effect of EQUIVALENCE, variables not in common block:

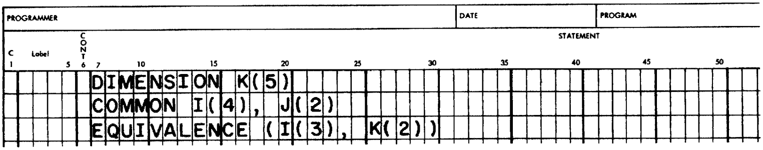

PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1
5
6
7
10
15
20
25
30
35
40
45
50

```
      DIMENSION I (4), J(2), K(5)
      EQUIVALENCE (I(3), K(2))
```

storage is assigned as follows:

Arrays		Quantities
I (1)		integer 1
I (2)	K (1)	integer 2
I (3)	K (2)	integer 3
I (4)	K (3)	integer 4
	K (4)	integer 5
	K (5)	integer 6
J (1)		integer 7
J (2)		integer 8

b) Effect of EQUIVALENCE, some variables in common block:

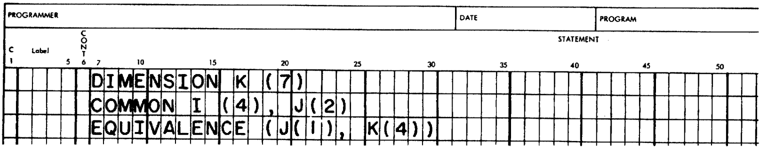

PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1
5
6
7
10
15
20
25
30
35
40
45
50

```
      DIMENSION K(5)
      COMMON I(4), J(2)
      EQUIVALENCE (I(3), K(2))
```

storage is assigned as follows:

Arrays		Quantities	
I (1)		integer 1	common block
I (2)	K (1)	integer 2	
I (3)	K (2)	integer 3	
I (4)	K (3)	integer 4	
J (1)	K (4)	integer 5	
J (2)	K (5)	integer 6	

c) Effect of EQUIVALENCE on the length of the common block:


PROGRAMMER DATE PROGRAM

C Label CONT STATEMENT

```
      DIMENSION K (7)
      COMMON I (4), J(2)
      EQUIVALENCE (J(1), K(4))
```

storage is assigned as follows:

Arrays		Quantities	
I (1)		integer 1	common block
I (2)	K (1)	integer 2	
I (3)	K (2)	integer 3	
I (4)	K (3)	integer 4	
J (1)	K (4)	integer 5	
J (2)	K (5)	integer 6	
	K (6)	integer 7	
	K (7)	integer 8	

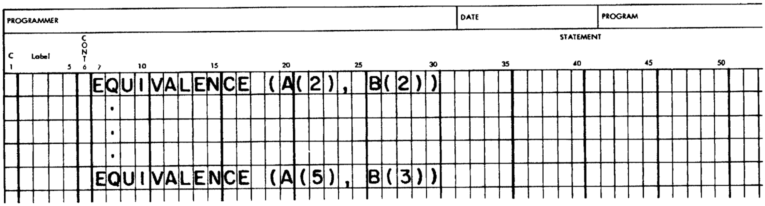
The value of the subscripts for an array being made equivalent to another array should not be such that the origin of the common block is changed (for example, EQUIVALENCE (I (3), K(4)).

	Arrays			Quantities
		K (1)	← origin changed	integer 1
origin →	I (1)	K (2)		integer 2
	I (2)	K (3)		integer 3
	I (3)	K (4)		integer 4
	I (4)	K (5)		integer 5
	J (1)	K (6)		integer 6
	J (2)	K (7)		integer 7

If contradictory EQUIVALENCE relationships are specified, a diagnostic message is printed.

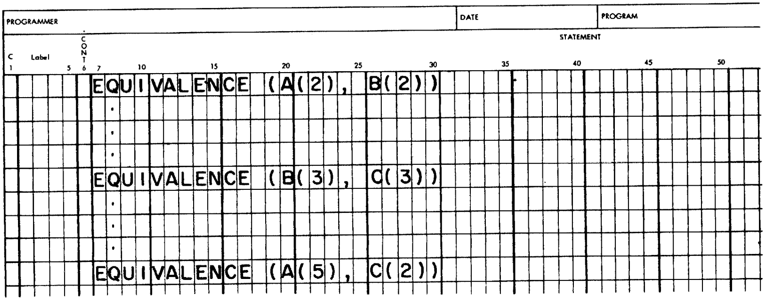
Example:

a)


PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1 5 6 7 10 15 20 25 30 35 40 45 50

```
      EQUIVALENCE (A(2), B(2))
      .
      .
      .
      EQUIVALENCE (A(5), B(3))
```

b)


PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1 5 6 7 10 15 20 25 30 35 40 45 50

```
      EQUIVALENCE (A(2), B(2))
      .
      .
      .
      EQUIVALENCE (B(3), C(3))
      .
      .
      .
      EQUIVALENCE (A(5), C(2))
```

CONTROL STATEMENTS 5

Program execution normally proceeds from statement to statement as they appear in the program. Control statements can be used to alter this sequence or cause a number of iterations of a program section. Control may be transferred to an executable statement only; a transfer to a non-executable statement will result in a program error which is usually recognized during compilation as a transfer to an undefined label.† With the DO statement, a predetermined sequence of instructions can be repeated a number of times with the stepping of a simple integer variable after each iteration.

Statements are labelled by unsigned numbers, 1 through 9999, which can be referred to from other sections of the program. A label up to four digits long precedes the FORTRAN statement and is separated from it by at least one blank or a zero. Imbedded blanks and leading zeros in the label are ignored: 1, 01, 0 1, 0001 are identical.

5.1 GO TO STATEMENTS

GO TO statements provide transfer of control.

GO TO k

This statement, an unconditional GO TO, causes the transfer of control to the statement labelled k .

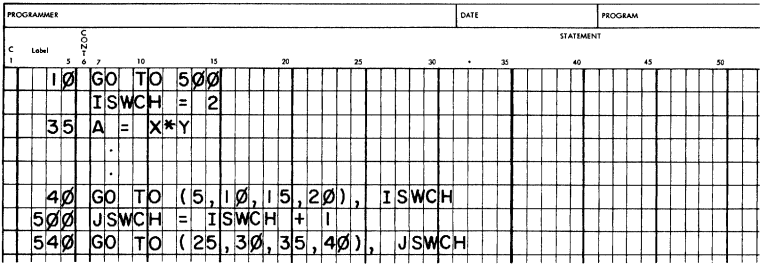
GO TO $(k_1, k_2, \ldots, k_n)$, i

This statement, a computed GO TO, acts as a many-branched transfer. The k's are statement labels and i is a simple integer variable. Execution of this statement causes the statement identified by the label k_j to be executed next, where j

† A transfer to a FORMAT statement is not detectable during compilation; if such an error occurs, no diagnostic message is produced.

is the value of i at the time of execution, and $1 \leq j \leq n$. If $i < 1$, a transfer to k_1 occurs; if $i > n$, a transfer to k_n occurs.

Examples:



PROGRAMMER | DATE | PROGRAM

STATEMENT

```
   1Ø GO TO 5ØØ
      ISWCH = 2
   35 A = X*Y
      .
      .
   4Ø GO TO (5,1Ø,15,2Ø), ISWCH
  5ØØ JSWCH = ISWCH + 1
  54Ø GO TO (25,3Ø,35,4Ø), JSWCH
```

At statement 40, control transfers to statement 10, which is an unconditional transfer to statement 500. At 540 control transfers to statement 35.

5.2 IF STATEMENTS

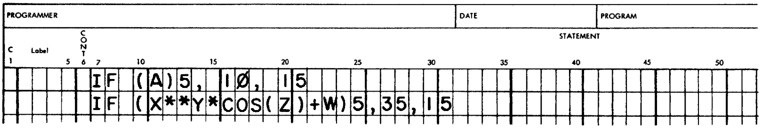
The arithmetic IF statement provides conditional transfer of control

IF (e) k_1, k_2, k_3

The e is an arithmetic expression and the k's are statement labels. The arithmetic IF is a three-way branch. Execution of this statement causes evaluation of the expression and transfer of control depending on the following conditions:

$e < 0$, go to k_1
$e = 0$, go to k_2
$e > 0$, go to k_3

Examples:



PROGRAMMER | DATE | PROGRAM

STATEMENT

```
      IF (A)5, 1Ø, 15
      IF (X**Y*COS(Z)+W)5,35,15
```

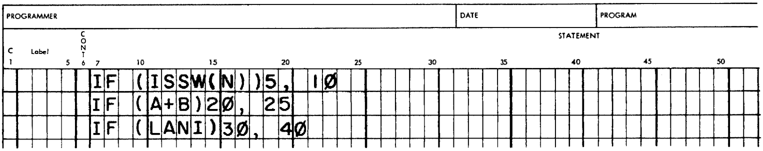
The logical IF statement provides conditional transfer of control to either of two statements:

IF (e) k_1, k_2

The e is an arithmetic expression that may yield a negative or non-negative (positive or zero) value. Execution of this statement causes evaluation of the expression and transfer of control under the following conditions:

$e < 0$, go to k_1
$e \geq 0$, go to k_2

<u>Examples:</u>


PROGRAMMER
DATE
PROGRAM
STATEMENT
C
Label
CONT
1
5
6
7
10
15
20
25
30
35
40
45
50

```
      IF (ISSW(N))5, 1Ø
      IF (A+B)2Ø, 25
      IF (LANI)3Ø, 4Ø
```

5.3 DO STATEMENTS

A DO statement makes it possible to repeat a group of statements.

DO n i = m_1, m_2, m_3

or

DO n i = m_1, m_2

The n is the label of an executable statement which ends the group of statements. The statement, called the terminal statement, must physically follow the DO statement in the source program. It may not be a GO TO of any form, IF, RETURN, STOP, PAUSE, or DO statement.

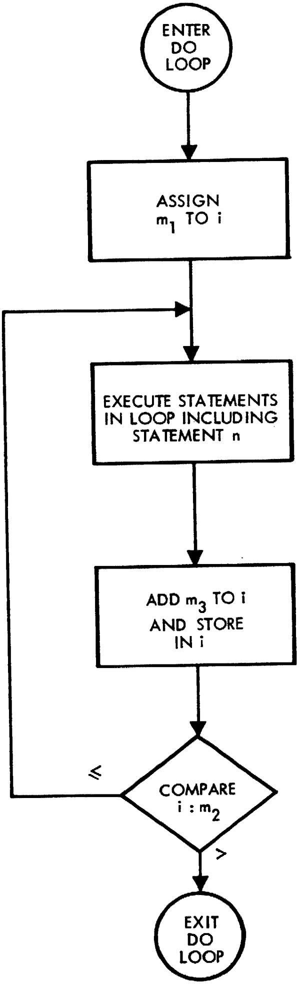
The i is the control variable; it may be a simple integer variable.

The m's are indexing parameters: m_1 is the initial parameter; m_2 , the terminal parameter; and m_3 , the incrementation parameter. They may be unsigned integer constants or simple integer variables. At time of execution, they all must be greater than zero. If m_3 does not appear (second form), the incrementation value is assumed to be 1.

A DO statement defines a loop. Associated with each DO statemeant is a range that is defined to be those executable statements following the DO, to and including the terminal statement associated with the DO. At time of execution, the following steps occur:

1. The control variable is assigned the value of the initial parameter.
2. The range of the DO is executed.
3. The terminal statement is executed and the control variable is increased by the value of the incrementation parameter.
4. The control variable is compared with the terminal parameter. If less than or equal to the terminal parameter, the sequence is repeated starting at step 2. If the control variable exceeds the terminal parameter, the DO loop is satisfied and control transfers to the statement following n . The control variable becomes undefined.

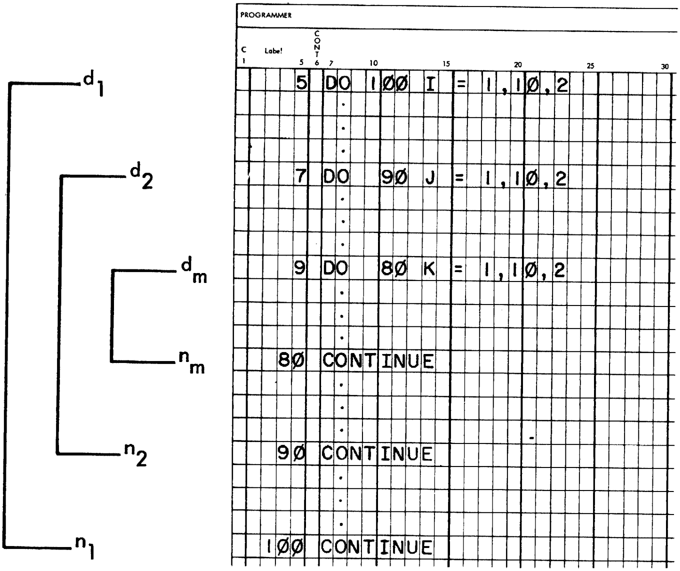
Should m_1 exceed m_2 on the initial entry to the loop, the range of the DO is executed and control passes to the statement after n . If a transfer out of the DO loop occurs before the DO is satisfied, the current value of the control variable is preserved. The control variable, initial parameters, terminal parameter, and incrementation parameters may not be redefined during the execution of the range of the DO loop.


ENTER DO LOOP
ASSIGN m_1 TO i
EXECUTE STATEMENTS IN LOOP INCLUDING STATEMENT n
ADD m_3 TO i AND STORE IN i
COMPARE i : m_2
≤
>
EXIT DO LOOP

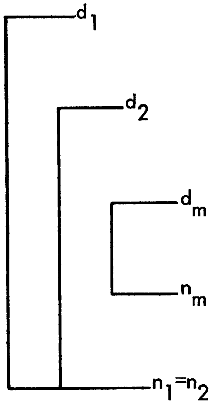
DO Nests

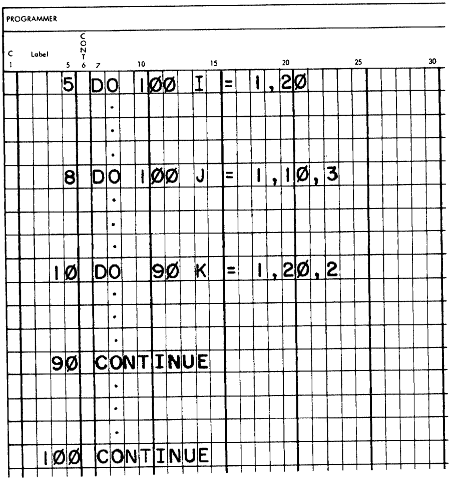
When the range of a DO loop contains another DO loop, the latter is said to be nested. DO loops may be nested 10 deep. The last statement of a nested DO loop must be the same as the last statement of the outer loop or occur before it. If $d_1, d_2, \ldots, d_n$ are DO statements, which appear in the order indicated by the subscripts; and if $n_1, n_2, \ldots, n_m$ are the respective terminal statements, then n_m must appear before or be the same as n_{m-1} , n_{m-1} must appear before or be the same as n_2, and n_2 must appear before or be the same as n_1.

Examples:

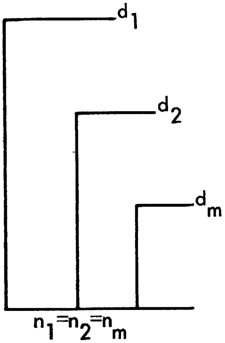


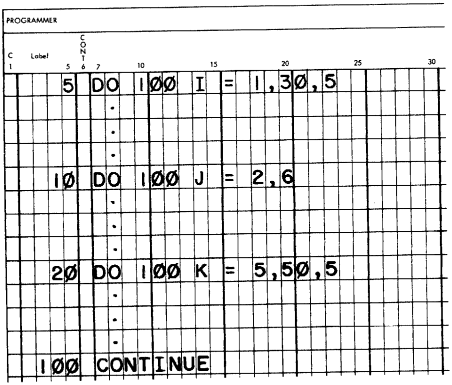
```
    5 DO 1ØØ I = 1,1Ø,2
      .
      .
      .
    7 DO  9Ø J = 1,1Ø,2
      .
      .
      .
    9 DO  8Ø K = 1,1Ø,2
      .
      .
      .
   8Ø CONTINUE
      .
      .
      .
   9Ø CONTINUE
      .
      .
      .
  1ØØ CONTINUE
```





```
PROGRAMMER
C  Label  CONT
    5 DO 1ØØ I = 1,2Ø
       .
       .
       .
    8 DO 1ØØ J = 1,1Ø,3
       .
       .
       .
   1Ø DO  9Ø K = 1,2Ø,2
       .
       .
       .
   9Ø CONTINUE
       .
       .
       .
  1ØØ CONTINUE
```



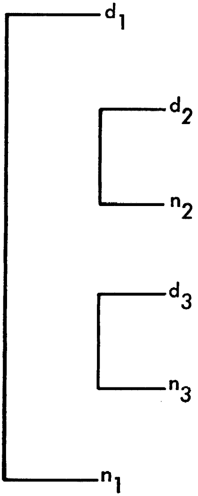


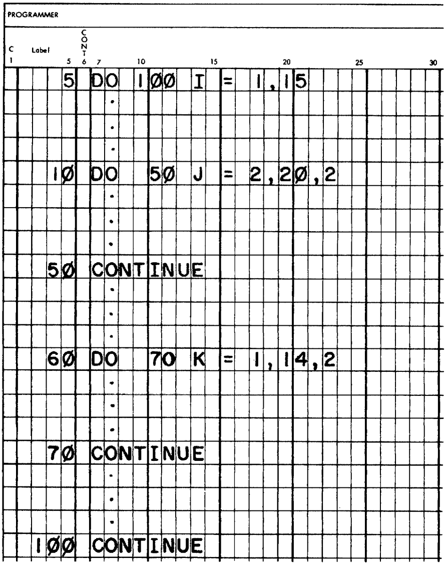
```
PROGRAMMER
C  Label  CONT
    5 DO 1ØØ I = 1,3Ø,5
       .
       .
       .
   1Ø DO 1ØØ J = 2,6
       .
       .
       .
   2Ø DO 1ØØ K = 5,5Ø,5
       .
       .
       .
  1ØØ CONTINUE
```

If one or more nested loops have the same terminal statement, when the inner DO is satisfied, the control variable for the next outer loop is incremented and tested against its associated terminal parameter. Control transfers to the statement following the terminal statement only when all related loops are satisfied.

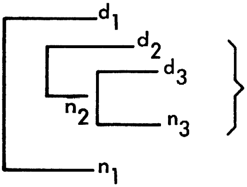
DO loops may be nested in common with other loops as long as their ranges do not overlap.

Examples:




PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20
25
30

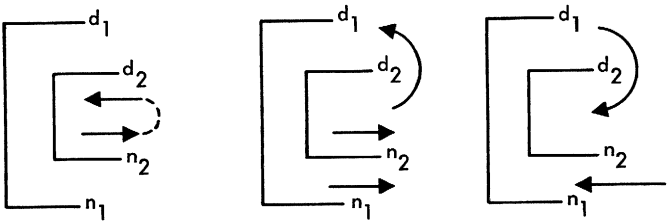
```
    5 DO 1ØØ I = 1,15
        .
        .
        .
   1Ø DO 5Ø  J = 2,2Ø,2
        .
        .
        .
   5Ø CONTINUE
        .
        .
        .
   6Ø DO 7Ø  K = 1,14,2
        .
        .
        .
   7Ø CONTINUE
        .
        .
        .
  1ØØ CONTINUE
```



Invalid, ranges overlap

In a DO nest, a transfer may be made from an inner loop into an outer loop, and transfer is permissible outside of the loop. It is illegal, however, for a GO TO or IF to initiate a transfer of control from outside of the range of a DO into its range.

When nested DO loops have the same terminal statement, a transfer to that terminal statement causes a transfer to the innermost logs of the nest. When this transfer occurs, the current value of the control variable for the innermost loop is incremented and that loop is executed until its range is satisfied, etc.



VALID TRANSFERS

INVALID TRANSFERS

5.4 CONTINUE

This statement acts as no-operation instruction.

```
CONTINUE
```

The CONTINUE statement is most frequently used as the last statement of a DO loop to provide a loop termination when a GO TO or IF would normally be the last statement of the loop. If used elsewhere in the source program, it acts as a do-nothing instruction and control passes to the next sequential program statement.

5.5 PAUSE

This statement provides a temporary program halt.

```
PAUSE n
   or
PAUSE
```

n may be up to four octal digits (without a B suffix) in the range 0 to 7777. This statement halts the execution of the program and types PAUSE on the Standard Output unit. The value of n , if given is displayed in the A-Register. Program execution resumes at the next statement.

5.6 STOP

The STOP statement terminates the execution of the program.

```
STOP n
   or
STOP
```

n may be up to four octal digits (without a B suffix) in the range 0 to 7777. This statement halts the execution of the program and types STOP on the Standard Output unit. The value of n , if given, is in the A-Register.

5.7 END

The END statement indicates the physical end of a program or subprogram. It has the form:

```
END name
```

The END statement is required for every program or subprogram. The name of the program can be included, but it is ignored by the compiler. The END statement is executable in the sense that it will effect return from a subprogram in the absence of a RETURN statement. An END statement may be labeled and may serve as a junction point.

5.8 END$

The END$ statement indicates the physical end of five or less programs or subprograms that are to be compiled at one time. If there are four or less programs, the statement is printed on the source program listing. If there are exactly five, the statement is not printed. If more than five programs are on the same tape, the END$ may be omitted after the fifth program; the compiler stops accepting input after the fifth is processed.

MAIN PROGRAM, FUNCTIONS, AND SUBROUTINES 6

A FORTRAN program consists of a main program with or without subprograms. Subprograms, which are either functions or subroutines, are sets of statements that may be written and compiled separately from the main program.

The main program calls or references subprograms; and subprograms may call or reference other subprograms as long as the calls are non-recursive. That is, if program A calls subprogram B, subprogram B may not call program A. Furthermore, a program or subprogram may not call itself. A calling program is a main program or subprogram that refers to another subprogram.

In addition to multi-statement function subprograms, a function may be defined by a single statement in the program (statement function) or it may be defined as basic external function. A statement function definition may appear in a main program or subprogram body and is available only to the main program or subprogram containing it. A statement function may contain references to function subprograms, basic external functions, or other previously defined statement functions in the same subprogram. Basic external function references may appear in the main program, subprogram, and statement functions.

Main programs, subprograms, statement functions, and basic external functions communicate by means of arguments (parameters). The arguments appearing in a subroutine call or function reference are actual arguments. The corresponding entities appearing with the subprogram, statement function, or basic external function definition are the dummy arguments.

6.1 ARGUMENT CHARACTERISTICS

Actual and dummy arguments must agree in order, type, and number. If they do not agree in type, errors may result in the program execution, since no conversion takes place and no diagnostic messages are produced.

Within subprograms, dummy arguments may be array names or simple variables; for statement functions, they may be variables only. Dummy arguments are local to the subprogram or statement function containing them and, therefore, may be the same as names appearing elsewhere in the program. A maximum of 63 dummy arguments may be used in a function or subroutine.

No element of a dummy argument list may appear in a COMMON or EQUIVALENCE statement within the subprogram. If it does, a compiler diagnostic results. When a dummy argument represents an array, it should be declared in a DIMENSION statement within the subprogram. If it is not declared, only the first element of the array will be available to the subprogram and the array name must appear in the subprogram without subscripts.

Actual arguments appearing in subroutine calls and function references may be any of the following:

A constant
A variable name
An array element name
An array name
Any other arithmetic expression

6.2 MAIN PROGRAM

The first statement of a main program may be the following:

PROGRAM name

The name is an alphanumeric identifier of up to five characters. If the PROGRAM statement is omitted, the compiler assigns the name "FTN."

6.3 SUBROUTINE SUBPROGRAM

An external subroutine is a computational procedure which may return none, one, or more than one value through its arguments or through common storage. No value or type is associated with the name of a subroutine.

The first statement of a subroutine subprogram gives its name and, if relevant, its dummy arguments.

SUBROUTINE s (a_1, a_2, . . . , a_n)

or

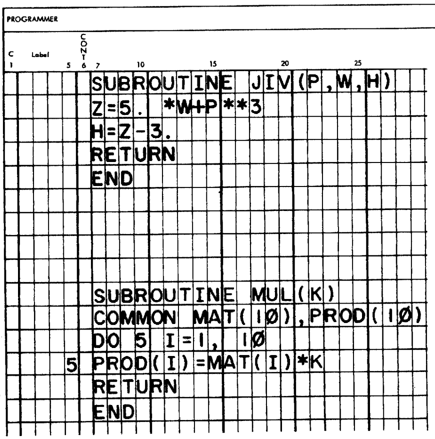
SUBROUTINE s

The symbolic name, s, is an alphanumeric identifier of up to five characters by which the subroutine is called. If the subroutine is unnamed the compiler will assign the name of "." (period). The a's are the dummy arguments of the subroutine.

The name of the subroutine must not appear in any other statement within the subprogram.

The subroutine may define or redefine one or more of its arguments and areas in common so as to effectively return results. It may contain any statements except FUNCTION, another SUBROUTINE statement, or any statement that directly or indirectly references the subroutine being defined. It must have at least one RETURN or END statement which returns control to the calling program.

Examples:


PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20
25

```
      SUBROUTINE JIV(P,W,H)
      Z=5. *W+P**3
      H=Z-3.
      RETURN
      END
```

P, W and H are the dummy parameters. Actual values supplied by a calling program are to be substituted for P and W. The variable name supplied for H would contain the result on return to the calling program.

```
      SUBROUTINE MUL(K)
      COMMON MAT(1Ø),PROD(1Ø)
      DO 5 I=1, 1Ø
    5 PROD(I)=MAT(I)*K
      RETURN
      END
```

MUL multiplies the array supplied for MAT by the single value supplied for K to produce values to be stored in array PROD.

6.4 SUBROUTINE CALL

The executable statement in the calling program for referring to a subroutine is:

CALL s (a_1, a_2,..., a_n)

or

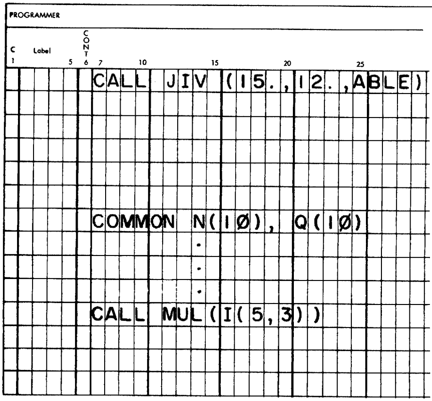
CALL s

The symbolic name, s, identifies the subroutine being called; the a's define the actual arguments. The name may not appear in any specification statements in the calling program.

If an actual argument corresponds to a dummy argument that is defined or redefined in the called subprogram, the actual argument must be a variable name, an array element name, or an array name.

The CALL statement transfers control to the subroutine. Execution of the subroutine results in an association of actual arguments with all appearances of dummy arguments in executable statement and function definition statements. If the actual argument is an expression, the association is by value rather than by name. Following these associations, the statements of the subprogram are executed. When a RETURN or END statement is encountered, control is returned to the next executable statement following the CALL in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until satisfied.

Examples:


PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20
25

```
      CALL JIV (15.,12.,ABLE)




      COMMON N(1Ø), Q(1Ø)
             .
             .
             .
      CALL MUL(I(5,3))
```

These calls provide actual arguments for the subroutines defined in the previous example. In subroutine JIV, 15. is substituted for P; 12., for W; and ABLE, for H.

For subroutine MUL, the data is passed via COMMON. The value supplied for the dummy argument K is element (5,3) of matrix I of the calling program.

6.5 FUNCTION SUBPROGRAM

A function subprogram is a computational procedure which returns a single value associated with the function name. The type of the function is determined by the name; an integer quantity is returned if the name begins with I, J, K, L, M, or N, otherwise it will be a real quantity.

The first statement of a function subprogram must have the following form:

FUNCTION f $(a_1, a_2, \ldots, a_n)$

The symbolic name, f, is an alphanumeric identifier of up to five characters by which the function is referenced. If the function is unnamed the compiler will assign the name of "." (period). The a's are the dummy arguments of the function.

The name of the function must not appear in any non-executable statement in the subprogram. It must be used in the subprogram, however, at least once as any of the following:

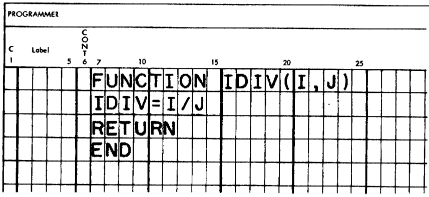
The left-hand identifier of an assignment statement
An element of an input list
An actual parameter of a subprogram reference

The value of name at the time of execution of a RETURN or END statement in the subprogram is called the value of the function.

The function subprogram may define or redefine one or more of its arguments and areas in common so as to effectively return results in addition to the value of the function. If the subprogram redefines variables contained in the same expression as the function reference, the evaluation sequence of the expression must be taken into account. Variables in the portion of the expression that is evaluated before the function reference is encountered and the values of variable subscripts are not affected by the execution of the function subprogram. Variables that appear following the function reference are modified according to the subprogram processing.

Examples:

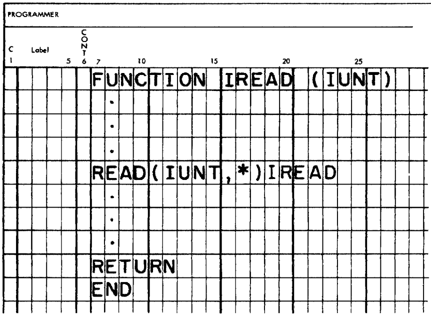
a)



```
      FUNCTION IDIV(I,J)
      IDIV=I/J
      RETURN
      END
```

The function IDIV calculates the value of I divided by J. On return to the calling program the result provided is the value of IDIV.

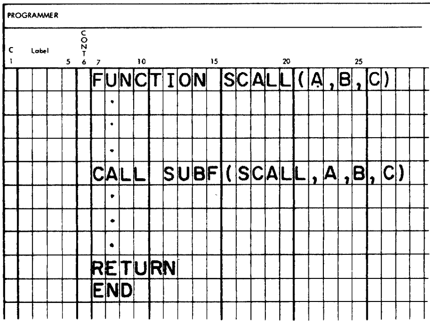
b)



```
      FUNCTION IREAD (IUNT)
       .
       .
       .
      READ(IUNT,*)IREAD
       .
       .
       .
      RETURN
      END
```

The function IREAD reads a value from the unit IUNT (specified as an actual parameter in the calling program.) IREAD has this value on return to the calling program.

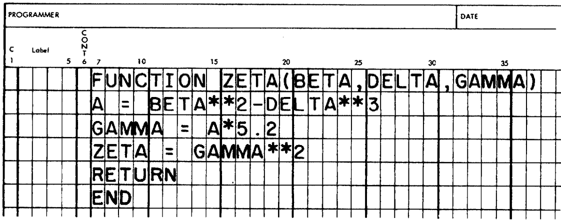
c)



```
      FUNCTION SCALL(A,B,C)
       .
       .
       .
      CALL SUBF(SCALL,A,B,C)
       .
       .
       .
      RETURN
      END
```

SCALL is both the function name and an actual parameter of a subroutine call. The value of SCALL is provided by SUBF and returned to the calling program.

d)



```
      FUNCTION ZETA(BETA,DELTA,GAMMA)
      A = BETA**2-DELTA**3
      GAMMA = A*5.2
      ZETA = GAMMA**2
      RETURN
      END
```

The function defines the value of GAMMA as well as finding the value of ZETA.

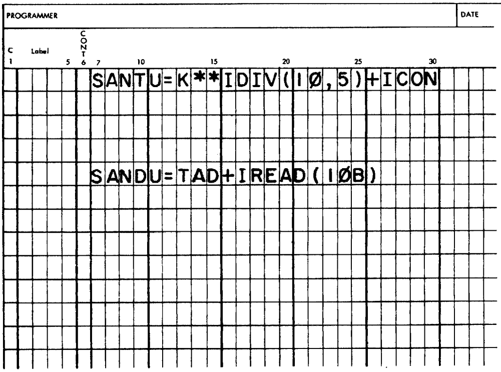
6.6 FUNCTION REFERENCE

A function subprogram is referenced by using the name and arguments in an arithmetic expression:

$$f\ (a_1, a_2, \ldots, a_n)$$

The type of function depends on the first letter of the name of the function referenced; the a's are the actual arguments. The reference may appear any place in an expression as an operand. The evaluated function will have a single value associated with the function name. When a function reference is encountered in an expression, control is transferred to the function indicated. Execution of the function results in an association of actual arguments with all appearances of dummy arguments in executable statements and function definition statements. If the actual argument is an expression, this association is by value rather than by name. Following these associations, the statements of the subprogram are executed. When a RETURN or END statement in the function subprogram is encountered, control returns to the statement containing the function reference. During execution the function also may define or redefine one or more of its arguments and areas in common.

Example:



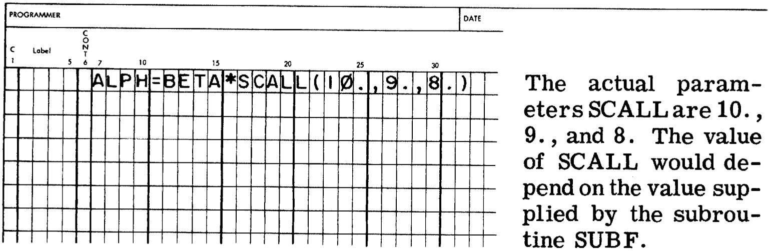
PROGRAMMER DATE

C Label CONT

```
a)       SANTU=K**IDIV(1Ø,5)+ICON

b)       SANDU=TAD+IREAD(1ØB)
```

The values of 10 and 5 are provided for I and J: The resulting value of IDIV would be 2. The function IREAD is called with 10B as the unit number. The value of IREAD would be the value of the item read from the device with unit reference number 10_8.

c)

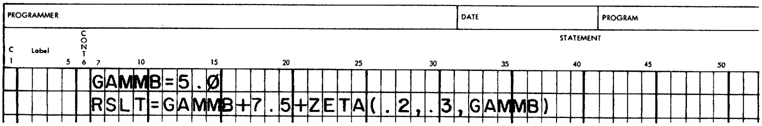


PROGRAMMER | DATE

```
      ALPH=BETA*SCALL(1Ø.,9.,8.)
```

The actual parameters SCALL are 10., 9., and 8. The value of SCALL would depend on the value supplied by the subroutine SUBF.

d) The program,



PROGRAMMER | DATE | PROGRAM

STATEMENT

```
      GAMMB=5.Ø
      RSLT=GAMMB+7.5+ZETA(.2,.3,GAMMB)
```

would result in the following calculation:

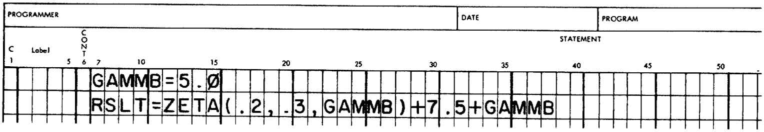
RSLT = 5.0 + 7.5 + ZETA

where ZETA would be determined as:

A = .2**2 - .3**3 = .04 - .027 = .013
GAMMA = .013*5.2 = .0676 (GAMMB is not altered)
ZETA = .0676**2 = .00456976

RSLT = 5.0 + 7.5 + .00456976
= 12.50456976

But, the program,



PROGRAMMER | DATE | PROGRAM

STATEMENT

```
      GAMMB=5.Ø
      RSLT=ZETA(.2,.3,GAMMB)+7.5+GAMMB
```

would result in the following calculations for ZETA and GAMMB:

A = .2**2 - .3**3 = .04 - .027 = .013
GAMMA = .013*5.2 = .0676 = GAMMB
ZETA = .0676**2 = .00456976

RSLT = .00456976 + 7.5 + .0676
= 7.57216976

When referring to a function which redefines an argument which appears as a variable elsewhere in the same expression, the order of evaluation (i. e., the order in which the expression is stated) is significant.

6.7 STATEMENT FUNCTION

A statement function is defined internally to the program or subprogram in which it is referenced and must precede the first executable statement. The definition is a single statement similar in form to an arithmetic assignment statement.

$$f\ (a_1,\ a_2, \ldots,\ a_n)\ =\ e$$

The name of the statement function, f , is an alphanumeric identifier; a single value is associated with the name. The dummy arguments, a's , must be simple variables. One to ten arguments may be used. The expression, e , may be an arithmetic expression and may contain references to basic external functions, previously defined statement functions, or function subprograms. The dummy arguments must appear in the expression. Other variables appearing in the expression have the same values as they have outside the statement function.

The statement function name must not appear in any specification statements in the program or subprogram containing it.

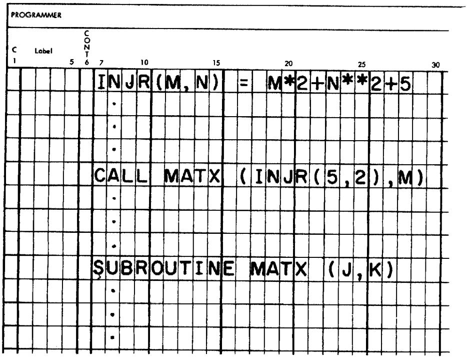
Statement functions must precede the first executable statement of the program or subprogram, but they must follow all specification statements.

A statement function reference has the form:

$$f\ (a_1,\ a_2, \ldots,\ a_n)$$

f is the function name and the a's are the actual arguments. A function reference with its appropriate actual arguments may be used to define the value of an actual argument in a subroutine call or function subprogram reference.

Example:



```
      INJR(M,N) = M*2+N**2+5
      .
      .
      .
      CALL MATX (INJR(5,2),M)
      .
      .
      .
      SUBROUTINE MATX (J,K)
      .
      .
      .
```

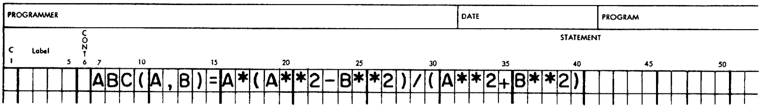
Statement function definition.

Subroutine call using statement function reference.

Execution of a statement function reference results in an association of actual argument values with the corresponding dummy arguments in the expression of the function definition, and evaluation of the expression. Following this, the resultant value is made available to the expression that contained the function reference and control is returned to that statement.

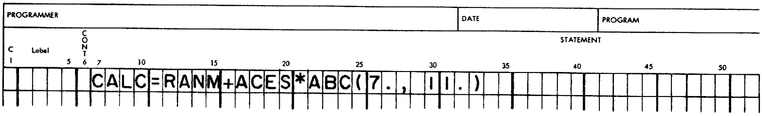
Example:

Statement function:



```
      ABC(A,B)=A*(A**2-B**2)/(A**2+B**2)
```

Function reference:



```
      CALC=RANM+ACES*ABC(7., 11.)
```

6.8 BASIC EXTERNAL FUNCTIONS

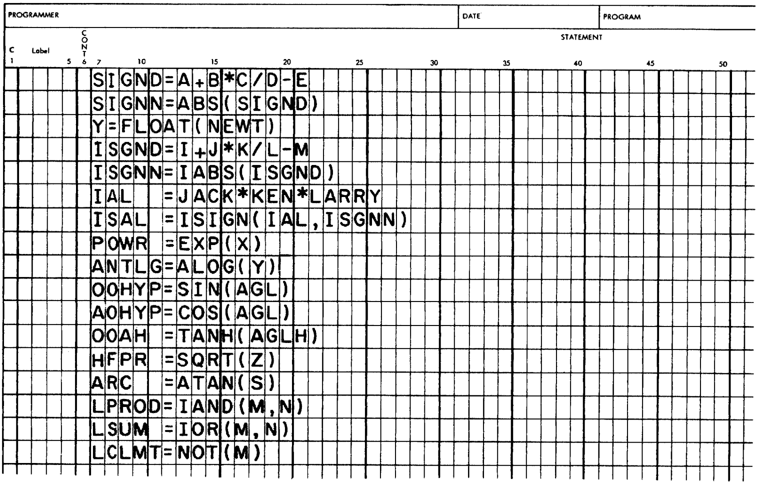
Certain basic functions are defined as part of the FORTRAN Library. When one of these appears as an operand in an expression, the compiler generates the appropriate calling sequence within the object program.

The types of these functions and their arguments are defined. The compiler recognizes the basic function and associates the type with the results. The actual arguments must correspond to the type required for the function; if not, a diagnostic message is issued. The functions available are shown below:

Function Name	Definition	Symbolic Name	No. of Arguments	Type of	
				Argument	Function
Absolute Value	$\|a\|$	ABS	1	Real	Real
		IABS	1	Integer	Integer
Float	Conversion from integer to real	FLOAT	1	Integer	Real
Fix	Conversion from real to integer	IFIX	1	Real	Integer
Transfer sign	Sign of a_2 times $\|a_1\|$	SIGN	2	Real	Real
		ISIGN	2	Integer	Integer
Exponential	e^a	EXP	1	Real	Real
Natural Logarithm	$\log_e (a)$	ALOG	1	Real	Real
Trigonometric Sine	sine (a)†	SIN	1	Real	Real
Trigonometric Cosine	cos (a)†	COS	1	Real	Real
Trigonometric Tangent	tan (a)†	TAN	1	Real	Real
Hyperbolic Tangent	tanh (a)	TANH	1	Real	Real
Square Root	$(a)^{1/2}$	SQRT	1	Real	Real
Arctangent	arctan(a)	ATAN	1	Real	Real
And (Boolean)	$a_1 \wedge a_2$	IAND	2	Integer	Integer
Or (Boolean)	$a_1 \vee a_2$	IOR	2	Integer	Integer
Not (Boolean)	$\neg a$	NOT	1	Integer	Integer
Sense Switch	Sense Switch Register switch (n)	ISSW	1	Integer	Integer

†a is in radians

Examples:


PROGRAMMER
DATE
PROGRAM
C
Label
CONT
STATEMENT
1 5 6 7 10 15 20 25 30 35 40 45 50

```
      SIGND=A+B*C/D-E
      SIGNN=ABS(SIGND)
      Y=FLOAT(NEWT)
      ISGND=I+J*K/L-M
      ISGNN=IABS(ISGND)
      IAL  =JACK*KEN*LARRY
      ISAL =ISIGN(IAL,ISGNN)
      POWR =EXP(X)
      ANTLG=ALOG(Y)
      OOHYP=SIN(AGL)
      AOHYP=COS(AGL)
      OOAH =TANH(AGLH)
      HFPR =SQRT(Z)
      ARC  =ATAN(S)
      LPROD=IAND(M,N)
      LSUM =IOR(M,N)
      LCLMT=NOT(M)
```

6.9 RETURN AND END

A subprogram normally contains a RETURN statement that indicates the end of logic flow within the subprogram and returns control to the calling program. It must always contain an END statement.

In function subprograms, control returns to the statement containing the function reference. In subroutine subprograms, control returns to the next executable statement following the CALL. A RETURN statement in the main program is interpreted as a STOP statement.

The END statement marks the physical end of a program, subroutine subprogram, or function subprogram. If the RETURN statement is omitted, END causes a return to the calling program. The END$ is required in addition to END statements when five or less subprograms are being compiled at one time.

INPUT/OUTPUT LISTS AND FORMAT CONTROL 7

Data transmission between internal storage and external equipment requires an input/output statement and, for ASCII character strings, either a FORMAT statement or format control symbols with the input data. The input/output statement specifies the input/output process, such as READ or WRITE; the unit of equipment on which the process is performed; and the list of data items to be moved. The FORMAT statements or control symbols provide conversion and editing information between the internal representation and the external character strings. If the data is in the form of strings of binary values, format control is unnecessary.

7.1 INPUT/OUTPUT LISTS

The input list specifies the names of the variables and array elements to which values are assigned on input. The output list specifies the references to the variables, array elements, and constants whose values are transmitted. The input and output lists are of the same form. The list elements consist of variable names, array elements, and array names separated by commas. The order in which the elements appear in the list is the sequence of transmission. If FORMAT statements are used, the order of the list elements must correspond to the order of the format descriptions for the data items. In array elements buffer length is limited to a maximum output of 60 computer words.

Subscripts in an input/output list may be of the form (exp$_1$, exp$_2$), where exp$_i$ is one of the following:

c*v+k
c*v-k
c*v
v+k
v-k
v
k

where c and k are integer constants and v is a simple integer variable previously defined or defined within an implied DO loop.

DO-Implied Lists

A DO-implied list consists of one or more list elements and indexing parameters. The general form is

$$(\ldots(\text{list, } i = m_1, m_2, m_3)\ldots)$$

list — Any series of arrays, array elements, or variables separated by commas

i — Control variable

m's — Index parameters in the form of unsigned integer constants or predefined integer variables

Data defined by the list elements is transmitted starting at the value of m_1 in increments of m_3 until m_2 is exceeded. If m_3 is omitted it is assumed to be one.

An implied DO loop may be used to transmit a simple variable or a sequence of variables more than one time.

Two-dimensional arrays may appear in the list with values specified for the range of the subscripts in an implied DO loop. The general form for an array is:

$$((a(d_1,d_2), i_1 = m_1, m_2, m_3), i_2 = n_1, n_2, n_3)$$

where,

a — An array name

d_1, d_2 — Subscripts of the array in one of the preceding forms

i_1, i_2 — Control variables representing either of the variable subscripts d_1 and d_2

m's, n's — Index parameters in the form of unsigned integer constants or predefined integer variables. If m_3 or n_3 is omitted, it is construed as 1.

The input/output list may contain nested implied DO loops. During execution, the control variables are assigned the values of the initial parameters ($i_1 = m_1$, $i_2 = n_1$). The first control variable defined in the list is incremented first. When the first control variable reaches the maximum value, it is reset; the next control variable to the right is incremented and the process is repeated until the last control variable has been incremented.

If the name of a dimensioned array appears in a list without subscripts, the entire array is transmitted.

Examples:

a) The DO-implied list:

```
   ((A(I,J), I=1, 20, 2), J=1, 50,5)
   replaces the following:
   DO x J=1, 50, 5
   DO x I=1, 20, 2
   transmit A (I,J)
x  CONTINUE
```

b) Other implied DO loops might be:

```
((ABLE(5*KID-3, 100*LID), KID=1,100), LID=1,10)
((A(I,J), I=1,5),J=1,5)  Transmit elements by column
((A(I,J),J=1,5), I=1,5)  Transmit elements by row.
```

c) Nested implied DO loops:

```
((((A(I,J),B(K,L),K=1,10),L=1,15),I=1,20),J=1,25)
 (((A(I,J),B(K),K=1,10),I=20,100,10),K=9,90,10)
```

d) Simple variable transmission:

```
(A, K=1, 10) Transmits 10 values of A.
```

e) Dimensioned array transmission:

```
   DIMENSION A(50,20)
           .
           .
           .
     ... A ...      list element
   is equivalent to:
   DO x I = 1,20
   DO x J = 1,50
   transmit A(J,I)
x  CONTINUE
```

7.2 FORMAT STATEMENT

ASCII input/output statements may refer to a FORMAT statement which contains the specifications relating to the internal-external structure of the corresponding input/output list elements.

FORMAT (spec$_1$, . . . , r(spec$_m$, . . .), spec$_n$, . . .)

The spec's are format specifications and r is an optional repetition factor which must be an unsigned integer constant. FORMAT specifications may be nested to a depth of one level. The FORMAT statement is non-executable and may appear anywhere in the program.

7.3 FORMAT STATEMENT CONVERSION SPECIFICATIONS

The data elements in the input/output lists may be converted from external to internal and from internal to external representation according to FORMAT conversion specifications.† FORMAT statements may also contain editing codes.

Conversion Specifications

rEw. d	Real number with exponent
rFw. d	Real number without exponent
rIw	Decimal integer
r@w	Octal integer
rKw	
rAw	Alphanumeric character

Editing Specification

nX	Blank field descriptor
nHh$_1$ h$_2$. . .h$_n$	Heading and labeling descriptors Specification should not be on more than one line. If continuation is necessary, specification should be broken up in two specifications.
r"h$_1$ h$_2$. . .h$_n$"	
r/	Begin new record

†If the type of a variable in the input/output list does not correspond to the type specified in the FORMAT statement, the formatter insures that the proper conversion from one type to the other will take place.

Both w and n are nonzero integer constants representing the width of the field in the external character string; n may be omitted if the width is one. d is an integer constant representing the number of digits in the fractional part of the string. r , the repeat count, is an optional nonzero integer constant indicating the number of times to repeat the succeeding basic field descriptor. Each h is one character.

Ew.d Output

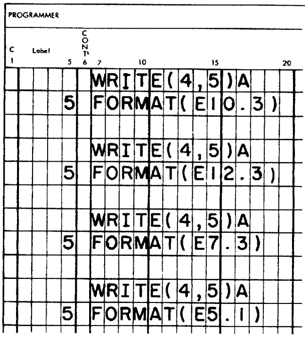
The E specification converts numbers in storage to character form for output. The field occupies w positions in the output record; the number appears in floating point form right justified in the field as:

$$\underline{\wedge}.x_1 \ldots x_d \; E\pm ee^{\dagger}$$

$x_1 \ldots x_d$ are the most significant digits of the value of the data to be output. ee are the digits in the exponent. Field w must be wide enough to contain significant digits, signs, decimal point, E , and exponent. Generally, w should be greater than or equal to d + 4.

If the field is not long enough to contain the output value, an attempt is made to adjust the value of d (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still too large for the field, dollar signs ($) are inserted in the entire field. If the field is longer than the output value, the quantity is right-justified with spaces to the left.

Examples:


PROGRAMMER
C
Label
CONT.
1
5
6
7
10
15
20

```
      WRITE(4,5)A
    5 FORMAT(E10.3)
```

A contains +12.34 or -12.34
Result is ∧∧.123E+02 or ∧-.123E+02

```
      WRITE(4,5)A
    5 FORMAT(E12.3)
```

A contains +12.34 or -12.34
Result is ∧∧∧∧.123E+02 or
∧∧∧-.123E+02

```
      WRITE(4,5)A
    5 FORMAT(E7.3)
```

A contains +12.34 or -12.34
Result is .12E+02 or -.1E+02

```
      WRITE(4,5)A
    5 FORMAT(E5.1)
```

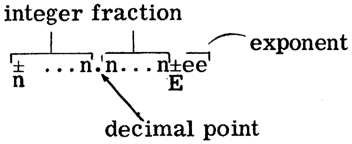
A contains +12.34
Result is $$$$$

[†] The caret symbol, ∧ , indicates the presence of a space.

Ew.d Input

The E specification converts the number in the input field (specified by w) to a real number and stores it in the appropriate storage locations.

The input field may consist of integer, fraction, and exponent subfields:



The integer subfield begins with a + or - sign, or a digit and may contain a string of digits terminated by a decimal point, an E , + , - , or the end of the input field.

The fraction subfield begins with a decimal point and may contain a string of digits terminated by an E , + , - , or the end of the input field.

The exponent field may begin with a sign or an E and contains a string of digits. When it begins with E , the + is optional between E and the string. The value of the string of digits should not exceed 38. The number may appear in any positions within the field; spaces in the field are ignored.

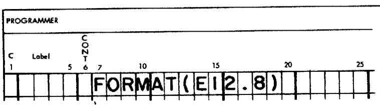
Examples:

```
+1.2345E2
123.456+9
-0.1234-6
.12345E-3
1234
+12345
+1234E6
```

When no decimal point is present in the input quantity, d acts as a negative power of ten scaling factor. The internal representation of the input quantity will be:

$$(\text{integer subfield})\ \mathrm{x}10^{-d}\ \mathrm{x}10^{(\text{exponent subfield})}$$

Example:



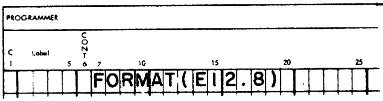
```
      FORMAT(E12.8)
```

Input quantity = ∧∧∧1234+5∧∧

Conversion performed: $1234\text{x}10^{-8}\text{x}10^{5}$
Result: 1.234

If a d value in the specification conflicts with the a decimal point appearing in an input field, the actual decimal point takes precedence.

Example:



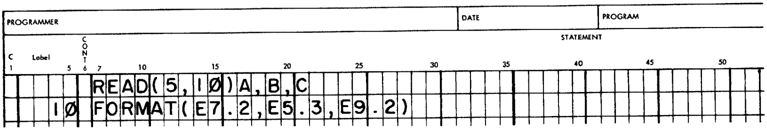
```
      FORMAT(E12.8)
```

Input quantity = ∧∧∧∧∧1.234+5

Quantity stored: $1.234\text{x}10^{5}$

The field width specified by w should always be the same as the width of the input field. When it is not, incorrect data may be read, converted and stored. The value of w should include positions for signs, the decimal point, the letter E , as well as the digits of the subfields:

Example:



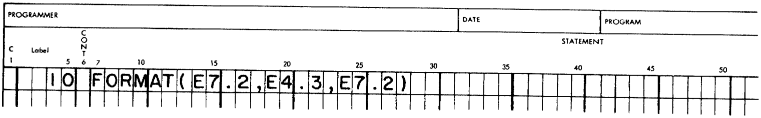
```
      READ(5,1Ø)A,B,C
   1Ø FORMAT(E7.2,E5.3,E9.2)
```

Assuming input data in contiguous fields:

```
-12.3E1+1234123.46E-3
|←— 7 —*—5 *——9 ——→|
```

The fields read would be:	and converted as:
-12.3E1	-123.
+1234	1.234
123.46E-3	.12346

However, if specifications were:



PROGRAMMER | DATE | PROGRAM

C | Label | CONT | STATEMENT

```
   10 FORMAT(E7.2,E4.3,E7.2)
```

The fields read would be:	and converted as:
-12.3E1	-123
+123	.123
4123.46	4123.46

The effects of possible FORMAT specification errors such as the above may not be detected by the system.

Examples:

FORMAT Specification	Input Field	Converted Value
E9.2	+1.2345E2	123.45
E9.4	-0.1234-6	-.0000001234
E4.2	1234	12.34

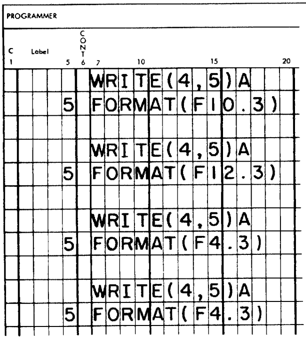
Fw.d Output

The F specification converts real numbers in storage to character form for output. The field occupies w positions and will appear as a decimal number, right justified in the field.

$\underline{\Delta}$x...x.x...x

The x's are the most significant digits. The number of decimal places to the right of the decimal point is specified by d. If d is zero, no digits appear to the right of the decimal point. The field must be wide enough to contain the significant digits, sign, and decimal point. If the number is positive, the + sign is suppressed. If the field is not long enough to contain the output value, an attempt is made to adjust the value of d (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still too large for the field, dollar signs ($) are inserted in the entire field. If the field is longer than the output value, the number is right-justified with spaces occupying the excess positions on the left.

Examples:



PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20

```
      WRITE(4,5)A
    5 FORMAT(F10.3)
```

A contains +12.34 or -12.34
Result: ∧∧∧∧12.340 or ∧∧∧-12.340

```
      WRITE(4,5)A
    5 FORMAT(F12.3)
```

A contains +12.34 or -12.34
Result: ∧∧∧∧∧∧12.340 or∧∧∧∧∧ -12.340

```
      WRITE(4,5)A
    5 FORMAT(F4.3)
```

A contains +12.34
Result: 12.3

```
      WRITE(4,5)A
    5 FORMAT(F4.3)
```

A contains +12345.12
Result: $$$$

Fw.d Input

The F specification input is identical to the E specification input. Although the fields are generally assumed to contain only a sign, integer, decimal point, and fraction; they may also contain an exponent subfield. All restrictions for Ew.d input apply.

Iw

The Iw specification converts internal values to output character strings, or input character strings to internal numbers. The output external field occupies w record positions and appears right justified (spaces on left) as:

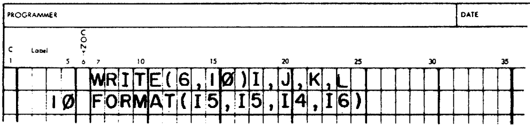
$$\overset{\wedge}{-}x_1 \ldots x_d$$

During input conversion, if a value is less than -32768_{10}, the value is converted to a positive 32767.

The x's represent the decimal digits (maximum of 5) of the integer. When the integer is positive on output, the sign is suppressed. If an output field is too short, dollar signs ($) will be placed in the output record.

The Iw specification, when used for input, is identical to an Fw.0 specification.

Examples:

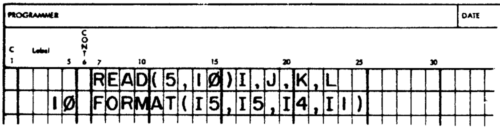

PROGRAMMER
DATE
C
Label
CONT
1 5 6 7 10 15 20 25 30 35

```
      WRITE(6,1Ø)I,J,K,L
   1Ø FORMAT(I5,I5,I4,I6)
```

I contains -1234
J contains +12345
K contains +12345
L contains +12345

Result: -123412345$$$$∧12345

|←5→|←5→|←4→|←6→|


PROGRAMMER
DATE
C
Label
CONT
1 5 6 7 10 15 20 25 30

```
      READ(5,1Ø)I,J,K,L
   1Ø FORMAT(I5,I5,I4,I1)
```

Input contains:

-∧12312∧∧3∧1∧23

|←5→|←5→|←4→|1|

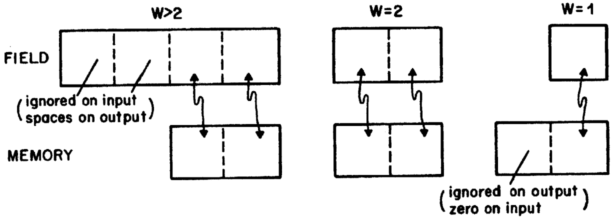
I contains -0123
J contains 12003
K contains 0102
L contains 3

Aw

This specification (not available in the 4K version of FORTRAN) causes alphanumeric data on an external medium to be translated to or from ASCII form in memory. The associated list element must be of type integer.

On input, if the field, as indicated by w, is greater than 2, the first w-2 characters are ignored; only the last two characters are read. When w equals 2, the two characters are read. If w equals 1, one character is read and stored in the right half of a computer word; zero is entered in the left half.

On output, if the field is greater than 2, two characters are written with right justification in the field; the leading positions are filled with spaces. If w equals 2, the two characters are written. If w equals 1, the character in the right half of the computer word is written.



Example:

```
        Input data:  AZZ213-ABCXABC137 - ZZ9 (CR) (LF)
        DIMENSION ID (5)
        READ (5, 1Ø) I2, I1, ID
10      FORMAT (A1Ø, A1, 5A2)
        Result:  I2  BC
                 I1  ØX
                 ID  AB
                     C1
                     37
                     -Z
                     Z9
```

r@w   rKw

Octal integer values are converted under either the @ or the K specification. The field is w octal digits in length; the corresponding list element must be of type integer. (Not available in the 4K version of FORTRAN.)

On input, if w is greater than or equal to 6, up to six octal digits are stored; non-octal digits appearing within the field are ignored. If the value of the octal digits within the field is greater than 177777, the results are unpredictable. If w is less than 6, or if less than six octal digits are encountered in the field, the number is right justified in the computer word with zero fill on the left.

On output, if the field is greater than 6, six octal digits are written with right justification in the field; the leading positions are filled with spaces. If w equals 6, the six octal digits are written. If w is less than 6, the w least significant octal digits are written.

Example:

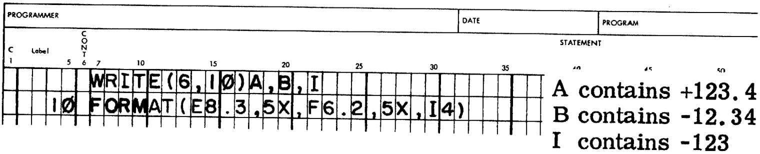
```
        Input data:  123456-1234562342342342, 396E-Ø5 (CR) (LF)
        DIMENSION ID(2), IE(2)
        READ (5, 10) IB, IC, ID, IE
10      FORMAT (@6, @7, 2@5, 2@4)
```

```
Result:  IB  123456
         IC  123456
         ID  Ø23423
             Ø42342
         IE  ØØØØ36
             ØØØØØ5
```

nX

The X specification may be used to include n blanks in an output record or to skip n characters on input to permit spacing of input/output quantities. In the specifications list, the comma following X is optional. ∧X is interpreted as 1X. 0X is not permitted.

Examples:



PROGRAMMER | DATE | PROGRAM

C Label CONT STATEMENT

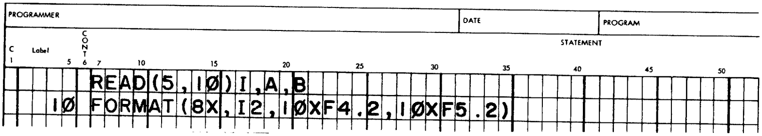
```
      WRITE(6,1Ø)A,B,I
   1Ø FORMAT(E8.3,5X,F6.2,5X,I4)
```

A contains +123.4
B contains -12.34
I contains -123

Result: ∧.1234E2∧∧∧∧∧-12.34∧∧∧∧∧-123

Input:

WEIGHT∧∧10∧∧PRICE∧∧$1.98∧∧TOTAL∧∧$19.80



PROGRAMMER | DATE | PROGRAM

C Label CONT STATEMENT

```
      READ(5,1Ø)I,A,B
   1Ø FORMAT(8X,I2,1ØXF4.2,1ØXF5.2)
```

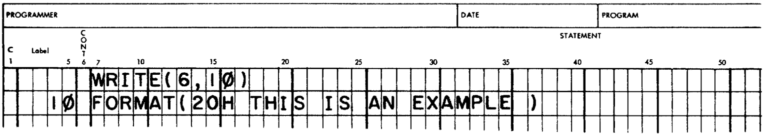
```
Result:  I  contains 10
         A  contains 1.98
         B  contains 19.80
```

$nHh_1h_2...h_n$

The H specification provides for the transfer of any combination of 8-bit ASCII characters, including blanks. n is an unsigned integer specifying the number of characters to the right of the H that are to be transmitted. The comma following the H specification is optional. ∧H is interpreted as 1H. 0H is not permitted. An H specification should not span more than one line. If continuation is necessary the H specification should be broken off in 2H specifications, one on each line.

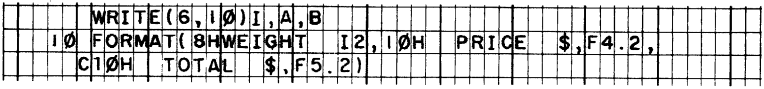
On output, the ASCII data in the FORMAT statement is written on the unit in the form of comments, titles, and headings.

Example:



```
      WRITE(6,1Ø)
   1Ø FORMAT(20H THIS IS AN EXAMPLE )
```

Result: THIS IS AN EXAMPLE



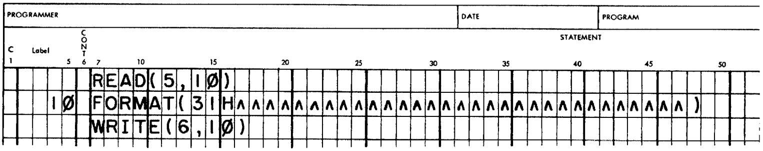
```
      WRITE(6,1Ø)I,A,B
   1Ø FORMAT(8HWEIGHT  I2,1ØH  PRICE  $,F4.2,
     C1ØH  TOTAL  $,F5.2)
```

I contains 10
A contains 1.98
B contains 19.80

Result: WEIGHT 10 PRICE $1.98 TOTAL $19.80

On input, the data is transmitted from the unit to the FORMAT statement. A subsequent output statement transfers the new data to the output record.

Examples:



```
      READ(5,1Ø)
   1Ø FORMAT(31H∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧)
      WRITE(6,1Ø)
```

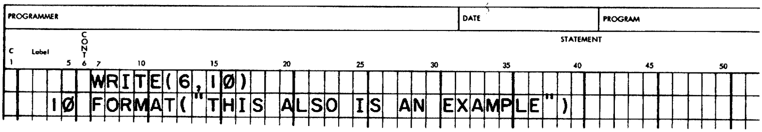
Input: H INPUT ALLOWS VARIABLE HEADERS

Result: H INPUT ALLOWS VARIABLE HEADERS

r"$h_1h_2...h_n$"

This specification also provides for the transfer of any combination of ASCII characters (except the quotation marks). The number of characters transmitted is the number of positions between the two quotation marks; field length is not specified. If r , an optional repeat count, is present, the character string within the quotation marks is repeated that number of times. Commas preceding the initial quotation mark and following the closing quotation are optional. As with H, the specification must be contained on one line.

Examples:

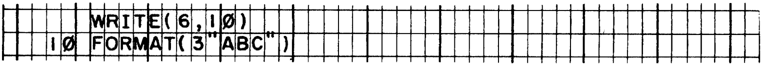


PROGRAMMER | DATE | PROGRAM

C Label CONT STATEMENT

```
      WRITE(6,1Ø)
   1Ø FORMAT("THIS ALSO IS AN EXAMPLE")
```

Result: THIS ALSO IS AN EXAMPLE



```
      WRITE(6,1Ø)
   1Ø FORMAT(3"ABC")
```

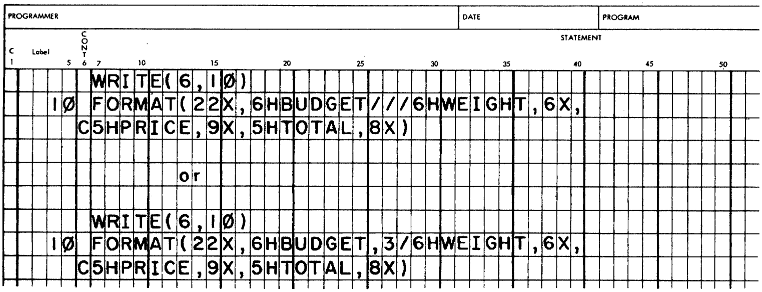
Result: ABCABCABC

On input, the number of characters within the quotation marks is skipped on the input field.

New Record

The slash, / , terminates the current record and signals the beginning of a new record of formatted data. It may occur anywhere in the specifications list and need not be separated from the other list elements by commas. Several records may be skipped by indicating consecutive slashes or by preceding the slash with a repetition factor; r-1 records are skipped for r/. On output the slash is used to skip lines, cards, or tape records; on input, it specifies that control passes to the next record or card.

Examples:



PROGRAMMER | DATE | PROGRAM

C Label CONT STATEMENT

```
      WRITE(6,1Ø)
   1Ø FORMAT(22X,6HBUDGET///6HWEIGHT,6X,
     C5HPRICE,9X,5HTOTAL,8X)

                 or

      WRITE(6,1Ø)
   1Ø FORMAT(22X,6HBUDGET,3/6HWEIGHT,6X,
     C5HPRICE,9X,5HTOTAL,8X)
```

Result:

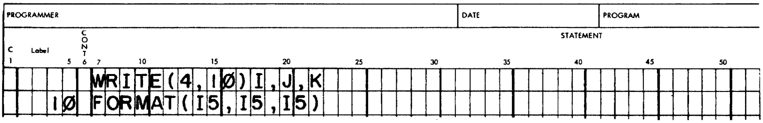
```
line 1    ^^^^^^^^ ^^^^^^^^^^ ^^^^ BUDGET

line 2

line 3

line 4    WEIGHT^^^^^^ PRICE^^^^^^^^^^ TOTAL^^^^^^^^
```

Repeat Specifications

Repetition of the field descriptors (except nH) is accomplished by preceding the descriptor with a repeat count, r . If the input/output list warrants, the conversion is interpreted repetitively up to the specified number of times.

Repetition of a group of field descriptors, including nH is accomplished by enclosing the group in parentheses and preceding the left parenthesis with a group repeat count. If no group repeat count is specified, a value of one is assumed. Grouped field descriptors may be nested to a depth of one level.

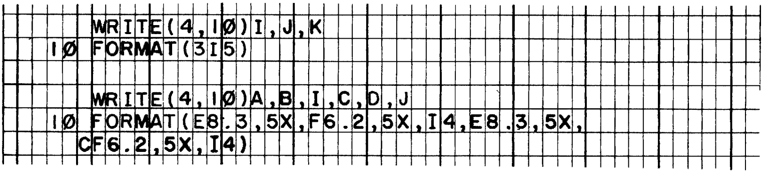
Examples:



PROGRAMMER | DATE | PROGRAM

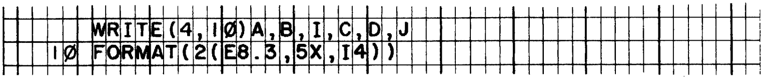
```
      WRITE(4,1Ø)I,J,K
   1Ø FORMAT(I5,I5,I5)
```

can be written as



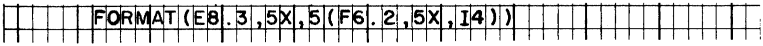
```
      WRITE(4,1Ø)I,J,K
   1Ø FORMAT(3I5)

      WRITE(4,1Ø)A,B,I,C,D,J
   1Ø FORMAT(E8.3,5X,F6.2,5X,I4,E8.3,5X,
     CF6.2,5X,I4)
```

can be written as



```
      WRITE(4,1Ø)A,B,I,C,D,J
   1Ø FORMAT(2(E8.3,5X,I4))
```

A nested repetition specification would be:



```
      FORMAT(E8.3,5X,5(F6.2,5X,I4))
```

The group F6.2, 5X, I4 would be written five times, and the entire group, once.

Unlimited Groups

FORMAT specifications may be repeated without use of the repetition factor. If list elements remain after all specifications in a FORMAT statement are processed, the rightmost group of repeated (enclosed in parentheses) specifications is used. If there is no repeated group, processing resumes with the first specification in the statement. On output, each time the rightmost parenthesis in the statement, or in the unlimited group, is reached, the current record is terminated.

7.4 FREE FIELD INPUT

By following certain conventions in the preparation of the input data, a 2116A FORTRAN program may be written without use of FORMAT statements. Special symbols included with the ASCII input data items direct the formatting:

space or,	Data item delimiters
/	Record terminator
+ -	Sign of item
. E + -	Floating point number
@	Octal integer
"..."	Comments

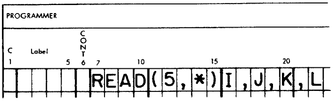
All other ASCII non-numeric characters are treated as spaces (and delimiters). Free field input may be used for numeric data only. Free field input is indicated in the FORTRAN READ statement by using an asterisk rather than a number of a FORMAT statement.

Data Item Delimiters

Any contiguous string of numeric and special formatting characters occurring between two commas, a comma and a space, or two spaces, is a data item whose value corresponds to a list element. A string of consecutive spaces is equivalent to one space. Two consecutive commas indicate that no data item is supplied for the corresponding list element; the current value of the list element is unchanged. An initial comma causes the first list element to be skipped.

Example:

1)

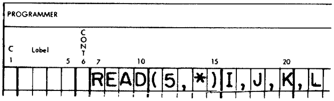

PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20

```
      READ(5,*)I,J,K,L
```

Input data: 1720, 1966
1980 1492

Result: I contains 1720
J contains 1966
K contains 1980
L contains 1492

2)


PROGRAMMER
C
Label
CONT
1
5
6
7
10
15
20

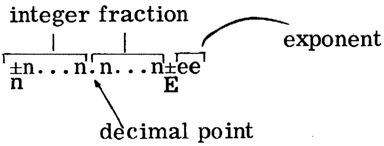
```
      READ(5,*)I,J,K,L
```

Input data: 1266,,1794,2000

Result: I contains 1266
J contains 1966
K contains 1794
L contains 2000

Floating Point Input

The symbols used to indicate a floating point data item are the same as those used in representing floating point data for FORMAT statement directed input:



If the decimal point is not present, it is assumed to follow the last digit.

Examples:

```
      READ(5,*)A,B,C,D,E
```

Input Data: 3.14, 314E-2, 3140-3, .0314+2, .314E1

All are equivalent to 3.14

Octal Input

An octal input item has the following format:

$$@\,x_1\ldots x_d$$

The symbol @ defines an octal integer. The x's are octal digits each in the range of 0 through 7. List elements corresponding to the octal data items must be type integer.

Record Terminator

A slash within a record causes the next record to be read immediately; the remainder of the current record is skipped.

Example:

```
      READ(5,*)II,JJ,KK,LL,MM
```

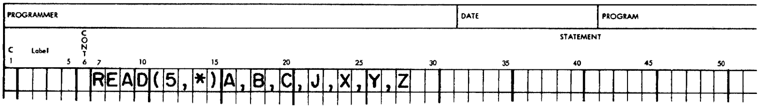
Input data: 987, 654, 321, 123/DESCENDING (CR) (LF)
456

Result:		
	II	contains 987
	JJ	contains 654
	KK	contains 321
	LL	contains 123
	MM	contains 456

List Terminator

If a line terminates (with a (CR) (LF)) and a slash has not been encountered, the input operation terminates even though all list elements may not have been processed. The current values of remaining elements are unchanged.

Examples:



PROGRAMMER | DATE | PROGRAM

C Label CONT STATEMENT

```
      READ(5,*)A,B,C,J,X,Y,Z
```

Input Data:

A=7.987 B=5E2 C=4.6859E-3 (CR) (LF)
J =3456 (CR) (LF)

Result:	
	A contains 7.987
	B contains 5E2
	C contains 4.6859E-3
	J, X, Y, Z are unchanged.

Comments

All characters appearing between a pair of quotation marks in the same line are considered to be comments and are ignored.

Examples:

"6.7321"	is a comment and ignored
6.7321	is a real number

INPUT/OUTPUT STATEMENTS 8

Input/output statements transfer information between memory and an external unit. The logical unit is specified as an integer variable that is defined elsewhere in the program or an integer constant.

Each statement may include a list of names of variables, arrays, and array elements. The named elements are assigned values on input and have their values transferred on output.

Records may be formatted or unformatted. A formatted record consists of a string of ASCII characters. The transfer of such a record requires the specification of a FORMAT statement or free field input data. An unformatted record consists of a string of binary values.

8.1 LOGICAL UNIT NUMBERS

FORTRAN input/output statements refer to logical unit numbers (1 to 63) whose meaning varies depending upon the operating system used. Refer to the appropriate manual. The operating system relates the logical unit number to a physical unit through system tables. Logical unit 4 always refers to a punch device, 5 to an input device, and 6 to a list output device.

8.2 FORMATTED READ, WRITE

A formatted READ statement is one of the forms:

```
READ (u, f)k
READ (u, *)k
READ (u, f)
```

Execution of this statement causes the input of the next ASCII records from unit u. The information is scanned and converted according to the FORMAT specification statement, f, and assigned to the elements of list k. If the input is free field, an asterisk is specified in the READ statement rather than the label of a FORMAT statement. If the list is absent, the FORMAT statement should contain editing specifications only.

A formatted WRITE statement may have one of the following forms:

```
WRITE (u, f)k
     or
WRITE (u, f)
```

This statement transfers ASCII information from locations given by names in the list k to output unit u. The values are converted and positioned as specified by the FORMAT statement f. If the list is absent, the FORMAT statement should contain editing specifications only.

8.3 UNFORMATTED READ, WRITE

An unformatted READ statement has one of the forms:

```
READ (u)k
     or
READ (u)
```

This statement transfers the next binary input record from the unit u to the elements of list k. The sequence of values required by the list may not exceed the sequence of values from the record. If no list is specified, READ (u) skips the next record.

An unformatted WRITE statement has the form:

```
WRITE (u)k
```

Execution of this statement creates the next record on unit u from the sequence of values represented by the list k.

8.4 AUXILIARY INPUT/OUTPUT STATEMENTS

There are three types of auxiliary input/output statements:

```
REWIND
BACKSPACE
ENDFILE
```

A REWIND statement has the form:

```
REWIND u
```

This statement causes the unit u to be positioned at its initial point. If the unit is currently at this position, the statement acts as a CONTINUE.

A BACKSPACE statement is as follows:

```
BACKSPACE u
```

BACKSPACE positions the unit u so that what had been the preceding record becomes the next record. If the unit is currently at its initial point, the statement acts as a CONTINUE.

An ENDFILE statement is of the form:

```
    ENDFILE u
```

Execution of this statement causes the recording of an end-of-file record on the output unit u. If given for an input unit, the statement acts as a CONTINUE.

COMPILER INPUT AND OUTPUT 9

The FORTRAN Compiler accepts as input, paper tape containing a control statement and a source language program. The output produced by the Compiler may include a punched paper tape containing the object program; a listing of the source language program with diagnostic messages, if any; and a listing of the object program in assembly level language.

9.1 CONTROL STATEMENT

The control statement must be the first statement of the source program; it directs the compiler.

FTN, p_1, p_2, p_3

FTN is a free field control statement. Following the comma are one to three parameters, in any order, which define the output to be produced. The control statement must be terminated by an end-of-statement mark, CR LF . Spaces embedded in the statement are ignored.

The parameters may be a combination of the following:

B Binary output: A program is to be punched in relocatable binary format suitable for loading by the Relocating Loader.

L List output: A listing of the source language program is to be produced as the source program is read in.

A Assembly listing: A listing of the object program in assembly level language is to be produced in the last pass.

T Symbol table only: A listing of the symbol table only is produced; in MTS, if both T and A are specified, only the last used will be decisive.

9.2 SOURCE PROGRAM

The source program follows the control statement. Each statement is followed by the end-of-statement mark, CR LF. Specifications statements must precede executable statements. The last statement in each program submitted for compilation must be an END statement. Up to five source programs may be compiled at one time. The last program must be followed by and END$ statement, if less than six programs are to be compiled.

The control statement, each of the five programs, and the END$ terminator may be submitted on a single tape or on separate tapes. If more than five programs are contained on a tape, the compiler processes the first five. The remaining programs must be compiled separately.

9.3 BINARY OUTPUT

The punch output produced by the compiler is a relocatable binary program. It does not include system subroutines introduced by the compiler, or library subroutines referred to in the program.

9.4 LIST OUTPUT

If the List Output parameter is specified, the first 72 characters of each line of the source program is printed on the List Output device. The END$ is the last statement printed. If exactly five programs are compiled, however, the END$ is omitted from the list.

If the Assembly listing parameter is specified, the program is printed in assembly level language on the List Output device. If the Symbol Table option is specified, the program listing is followed by a Symbol Table for the assembly level program.

The format for the assembly level listing is as follows:

Columns	Content
1-5	Zero-relative location (octal) of the instruction
6-7	Blank

Columns	Content
8-13	Object code word in octal
14	Relocation or external symbol indicator
15	Blank
16-18	Mnemonic operation code
19	Blank
20-25	Operand address in octal or external symbol name.
26-27	The indicator ",I" if indirect addressing is used.

The Symbol Table listing has the following format:

Columns	Content
1-5	Symbol, statement label, or numeric symbol assigned by the compiler.
6	Blank
7	Relocation indicator
8	Blank
9-14	The zero-relative value of the symbol

The characters that designate an external symbol or type of relocation for the operand address or a symbol in the Symbol Table are:

Character	Relocation Base
Blank	Absolute
R	Program relocatable
X	External symbol
C	Common relocatable

NOTE: The operating procedures for the FORTRAN Compiler are contained in the ASSEMBLER Appendix D, SIO SUBSYSTEMS OPERATION (5951-1390).

HP CHARACTER SET A

ASCII CHARACTER FORMAT

b_4	b_3	b_2	b_1								
			b_7	0	0	0	0	1	1	1	1
			b_6	0	0	1	1	0	0	1	1
			b_5	0	1	0	1	0	1	0	1
0	0	0	0	NULL	DC$_0$	ƀ	0	@	P	UNASSIGNED	UNASSIGNED
0	0	0	1	SOM	DC$_1$	!	1	A	Q		
0	0	1	0	EOA	DC$_2$	"	2	B	R		
0	0	1	1	EOM	DC$_3$	#	3	C	S		
0	1	0	0	EOT	DC4 (STOP)	$	4	D	T		
0	1	0	1	WRU	ERR	%	5	E	U		
0	1	1	0	RU	SYNC	&	6	F	V		
0	1	1	1	BELL	LEM	' (APOS)	7	G	W		
1	0	0	0	FE$_0$	S$_0$	(	8	H	X		
1	0	0	1	HT / SK	S$_1$	)	9	I	Y		
1	0	1	0	LF	S$_2$	*	:	J	Z		
1	0	1	1	V$_{TAB}$	S$_3$	+	;	K	[		
1	1	0	0	FF	S$_4$	, (COMMA)	<	L	\		ACK
1	1	0	1	CR	S$_5$	—	=	M	]		①
1	1	1	0	SO	S$_6$	.	>	N	↑		ESC
1	1	1	1	SI	S$_7$	/	?	O	←		DEL

Standard 7-bit set code positional order and notation are shown below with b_7 the high-order and b_1 the low-order, bit position.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
EXAMPLE: The code for "R" is:	1	0	1	0	0	1	0

LEGEND

NULL	Null/Idle
SOM	Start of message
EOA	End of address
EOM	End of message
EOT	End of transmission
WRU	"Who are you?"
RU	"Are you...?"
BELL	Audible signal
FE$_0$	Format effector
HT	Horizontal tabulation
SK	Skip (punched card)
LF	Line feed
V$_{TAB}$	Vertical tabulation
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DC$_0$	Device control reserved for data link escape

DC$_1$-DC$_3$	Device Control
DC$_4$(Stop)	Device control (stop)
ERR	Error
SYNC	Synchronous idle
LEM	Logical end of media
S$_0$-S$_7$	Separator (information)
ƀ	Word separator (space, normally non-printing)
<	Less than
>	Greater than
↑	Up arrow (Exponentiation)
←	Left arrow (Implies/Replaced by)
\	Reverse slant
ACK	Acknowledge
①	Unassigned control
ESC	Escape
DEL	Delete/Idle

ASSEMBLY LANGUAGE SUBPROGRAMS B

A FORTRAN program can refer to a subprogram that has been prepared using Assembler source language. The subprogram may be treated as a subroutine or as a function. The object code programs generated by FORTRAN and by the Assembler are then linked together by the Relocating Loader when the programs are loaded.

FORTRAN REFERENCE

In the FORTRAN program, a subroutine is called using the following statement:

CALL s $(a_1, a_2, \ldots, a_n)$

The symbolic name, s, identifies the subroutine and the a's are the actual arguments.

If the subprogram is a function, it is referenced by using the name and the actual arguments in an arithmetic expression:

$f(a_1, a_2, \ldots, a_n)$

As a result of either the call or the reference, FORTRAN generates the following coding sequence:

JSB s/f	Transfers control to subroutine or function
DEF*+n+1	Defines return location
DEF a_1	Defines address of a_1
DEF a_2	Defines address of a_2
⋮	
DEF a_n	Defines address of a_n

The words defining the addresses of the arguments may be direct or indirect depending on the actual arguments. For example, an integer constant as an actual argument would yield a direct reference; an integer variable might yield an indirect reference.

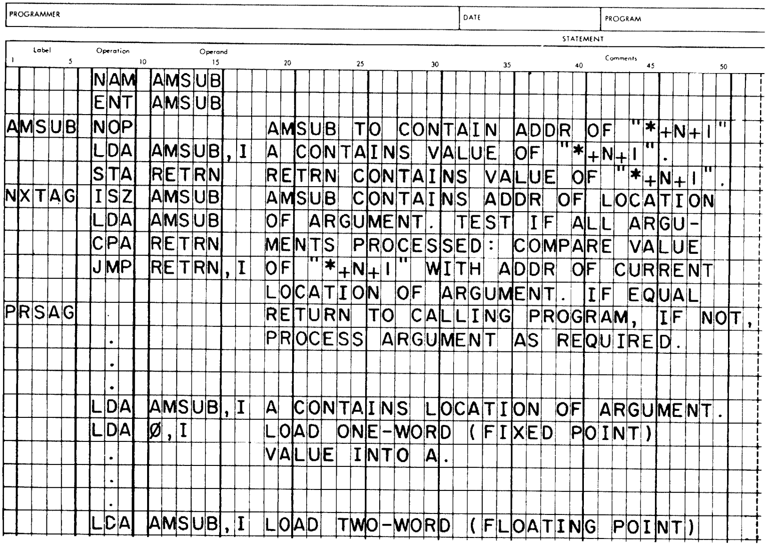
If the subprogram being referenced is a subroutine, it may return none, one, or more than one value through its arguments or through common storage. If the subprogram is a function, it is assumed to return a single value in the accumulators: a function of type integer returns a value in the A-Register; a function of type real returns a value in the A- and B-Registers.

The subprogram may transfer values directly by accessing the words in the calling sequence or it may make use of the FORTRAN library subroutine .ENTR to aid in the transfer.

DIRECT TRANSFER OF VALUES

Any suitable technique may be used to obtain or deliver values for the arguments and to return control to the calling program. If address arithmetic is used in conjunction with an argument (e.g., to process elements of an array), the base location must be a direct reference; the location given in the calling sequence must be checked to determine if it is a direct or indirect reference. If it is an indirect reference the location to which it points must also be checked, and so forth.

Example:



PROGRAMMER
DATE
PROGRAM
STATEMENT
Label
Operation
Operand
Comments
1
5
10
15
20
25
30
35
40
45
50

```
      NAM AMSUB
      ENT AMSUB
AMSUB NOP           AMSUB TO CONTAIN ADDR OF "*+N+1"
      LDA AMSUB,I   A CONTAINS VALUE OF "*+N+1".
      STA RETRN     RETRN CONTAINS VALUE OF "*+N+1".
NXTAG ISZ AMSUB     AMSUB CONTAINS ADDR OF LOCATION
      LDA AMSUB     OF ARGUMENT. TEST IF ALL ARGU-
      CPA RETRN     MENTS PROCESSED: COMPARE VALUE
      JMP RETRN,I   OF "*+N+1" WITH ADDR OF CURRENT
                    LOCATION OF ARGUMENT. IF EQUAL
PRSAG  .            RETURN TO CALLING PROGRAM, IF NOT,
       .            PROCESS ARGUMENT AS REQUIRED.
       .
       .
      LDA AMSUB,I   A CONTAINS LOCATION OF ARGUMENT.
      LDA Ø,I       LOAD ONE-WORD (FIXED POINT)
       .            VALUE INTO A.
       .
       .
      LCA AMSUB,I   LOAD TWO-WORD (FLOATING POINT)
      DLD Ø,I      VALUE INTO A AND B.
       .
       .
       .
      LDA AMSUB,I  STORE ONE-WORD VALUE IN ARGUMENT
      STA OUTAD    LOCATION.
      LDA W1VAL
      STA OUTAD,I
       .
       .
       .
      LDA AMSUB,I  STORE TWO-WORD IN ARGUMENT
      STA OUTAD    LOCATIONS.
      DLD W2VAL
      DST OUTAD,I
       .
       .
       .
      LDA AMSUB,I  A CONTAINS ADDR OF LOCATION OF
      SSA          ARGUMENT. TO DETERMINE IF REF IS
      JMP *+2      INDIRECT, TEST BIT 15. IF ONE,
      JMP *+5      SET TO ZERO WITH AND, THEN LOAD
      AND ANMSK    A WITH REFERENCED LOCATION.
      LDA Ø,I      REPEAT TEST WITH NEXT REF. WHEN
      JMP *-5      DIRECT REF ENCOUNTERED, PROCEED
ANMSK OCT Ø77777   WITH PROCESSING.
       .
       .
       .
      JMP NXTAG    RETURN THROUGH HERE WHEN NEXT
RETRN BSS 1        ARGUMENT IS REQUIRED.
OUTAD BSS 1
W1VAL BSS 1
W2VAL BSS 2
      END
```

The preceding example assumes that each argument is processed or partially processed before the next is obtained or delivered. Control returns to the calling program when all arguments have been picked up or delivered.

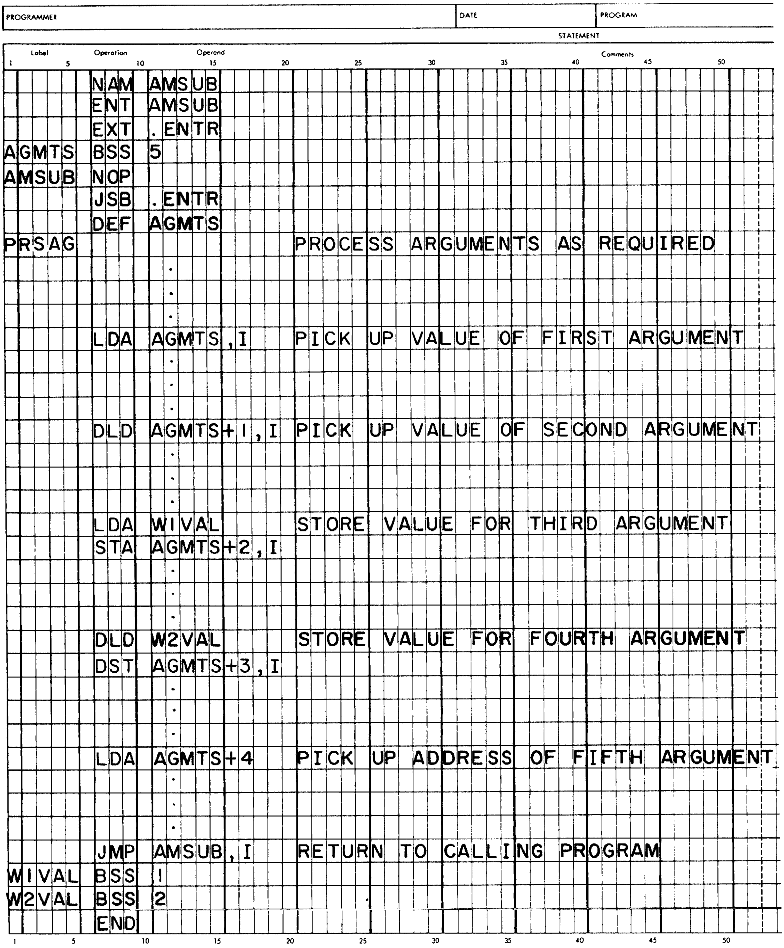
TRANSFER VIA .ENTR

The transfer of values to or from the locations listed in the calling sequence may be facilitated through use of the FORTRAN library subroutine .ENTR. This subroutine moves the addresses of the arguments into an area reserved within the Assembly language subroutine. The addresses stored in the reserved area are all direct references; .ENTR performs all the necessary direct/indirect testing, etc. It also sets the correct return address in the entry point location.

The general form of the subroutine is:

```
        NAM s              The subroutine name is s.
        ENT s
        EXT .ENTR          .ENTR must be declared as external.
a       BSS n              Reserves n words of storage for the
s       NOP                addresses of the arguments; this pseudo
                           instruction must directly precede the
                           entry point location,  s .

        JSB .ENTR
        DEF a              Defines first location of area used to
  (First instruction)      store argument addresses.
             .
             .
             .
        JMP s, I
        END
```

Example:



PROGRAMMER DATE PROGRAM

STATEMENT

```
      NAM AMSUB
      ENT AMSUB
      EXT .ENTR
AGMTS BSS 5
AMSUB NOP
      JSB .ENTR
      DEF AGMTS
PRSAG                   PROCESS ARGUMENTS AS REQUIRED
           .
           .
           .
      LDA AGMTS,I       PICK UP VALUE OF FIRST ARGUMENT
           .
           .
           .
      DLD AGMTS+1,I     PICK UP VALUE OF SECOND ARGUMENT
           .
           .
           .
      LDA W1VAL         STORE VALUE FOR THIRD ARGUMENT
      STA AGMTS+2,I
           .
           .
           .
      DLD W2VAL         STORE VALUE FOR FOURTH ARGUMENT
      DST AGMTS+3,I
           .
           .
           .
      LDA AGMTS+4       PICK UP ADDRESS OF FIFTH ARGUMENT
           .
           .
           .
      JMP AMSUB,I       RETURN TO CALLING PROGRAM
W1VAL BSS 1
W2VAL BSS 2
      END
```

SAMPLE PROGRAM C

Using Simpson's rule, calculate the value of the integral:

$$\int_a^b \frac{\cos x}{x}\, dx$$

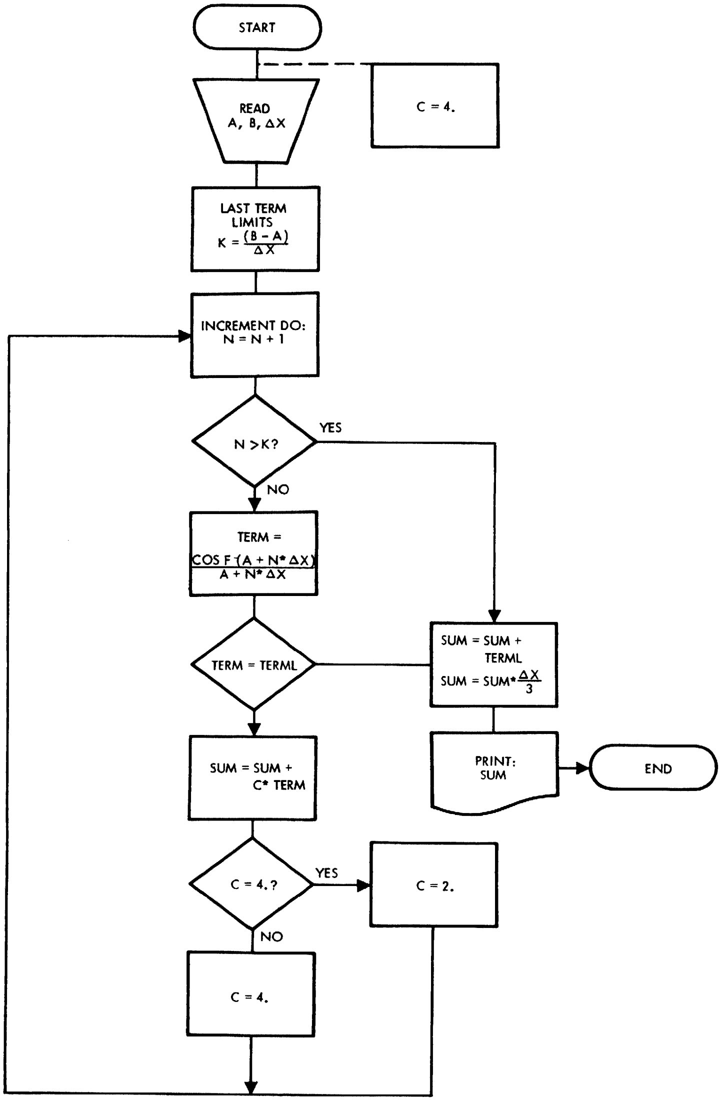
for the following possible values:

Variable	Range of Values
a	-6.99 to +6.99
b	-6.99 to +6.99
Δx	-.25 to +.25

Simpson's rule for approximating a definite integral is:

$$\int_a^b f(x)dx = \frac{\Delta x}{3}(f(a)+4f(a+\Delta x)+2f(a+2\Delta x)+4f(a+3\Delta x)+\ldots +f(b)$$

The last term is reached when $(a+k\Delta x)=b$, and when neither a 2 nor a 4 appears in front of the first or last term.



SAMPLE PROGRAM FLOWCHART

PROGRAMMER | DATE | PROGRAM

STATEMENT

C Label CONT

```
FTN,B,L,A
      PROGRAM SMPSN
      READ(1,1Ø) A,B,DELTX
   1Ø FORMAT(2E8.2,E7.2)
      TERML=COS(B)/B
      SUM=COS(A)/A
      K=(B-A)/DELTX
      C=4.
      I=K+1
      DO 6Ø N=1,I
      FN=N
      IF(N-K)2Ø,2Ø,7Ø
   2Ø TERM=COS(A+FN*DELTX)/(A+FN*DELTX)
      IF(TERM-TERML)3Ø,7Ø,3Ø
   3Ø SUM=SUM+C*TERM
      IF(C-4.)5Ø,4Ø,5Ø
   4Ø C=2.
      GO TO 6Ø
   5Ø C=4.
   6Ø CONTINUE
   7Ø SUM=SUM+TERML
   8Ø SUM=(SUM*DELTX)/3.
      WRITE(2,9Ø) SUM
   9Ø FORMAT("SUM=",E8.2)
      STOP
      END
      END$
```

M0251

Ø = ZERO   O = ALPHA O   I OR 1 = ONE   I = ALPHA I   LINE TERMINATED BY RETURN / LINE FEED (R/LF)
2 = TWO   Z = ALPHA Z   LINE IS DELETED BY RUBOUT BEFORE R/LF

OBJECT PROGRAM

Input and Output Data

```
    1.23    4.72     .25
SUM=-.63E+00
STOP
    1.23    2.01     .10
SUM=-.12E-01
STOP
    0.34    1.01     .02
SUM= .88E+00
STOP
    0.00    1.00     .01
SUM= .57E+36
STOP
    1.00    1.25     .05
SUM= .92E-01
STOP
```

FORTRAN ERROR MESSAGES D

During the compilation or assembly of programs, error messages are typed on the list output device to aid the programmer in debugging programs. Errors detected in the source program are indicated by a numeric code inserted before or after the statement in the List Output.

The format is as follows:

E-eeee: ssss + nnnn

eeee The error diagnostic code shown below.

ssss The statement label of the statement in which the error was detected. If unlabeled, ØØØØ is typed.

nnnn Ordinal number of the erroneous statement following the last labeled statement. (Comment statements are not included in this count.)

Error Code — Description

ØØØ1 Statement label error:

a) The label is in positions other than 1-5.

b) A character in the label is not numeric.

c) The label is not in the range 1-9999.

d) The label is doubly defined.

e) The label indicated is used in a GO TO, DO, or IF statement or in an I/O operation to name a FORMAT statement, but it does not appear in the label field for any statement in the program (printed after END).

Error Code — Description

ØØØ2 Unrecognized Statement

a) The statement being processes is not recognized as a valid statement.

b) A specifications statement follows an executable statement.

c) The specification statements are not in the following order:

DIMENSION
COMMON
EQUIVALENCE

d) A statement function precedes a specification statement.

ØØØ3 Parenthesis error: There are an unequal number of left and right parentheses in a statement.

ØØØ4 Illegal character or format:

a) A statement contains a character other than A Z, Ø through 9, or space =+–/(),.$”.

b) A statement does not have the proper format.

c) A control statement is missing, misspelled, or does not have the proper format.

d) An indexing parameter of a DO-loop is not an unsigned integer constant or simple integer variable or is specified as zero.

ØØØ5 Adjacent operators: An arithmetic expression contains adjacent arithmetic operators.

ØØØ6 Illegal subscript: A variable name is used both as a simple variable and a subscripted variable.

Error Code — Description

ØØØ7 Doubly defined variable:

a) A variable name appears more than once in a COMMON statement.

b) A variable name appears more than once in a DIMENSION statement.

c) A variable name appears more than once as a dummy argument in a statement function.

d) A program subroutine, or function name appears as a dummy parameter; in a specifications statement of the subroutine or function; or as a simple variable in a program or subroutine.

ØØØ8 Invalid parameter list:

a) The dummy parameter list for a subroutine or function exceeds 63.

b) Duplicate parameters appear in a statement function.

ØØØ9 Invalid arithmetic expression:

a) Missing operator

b) Illegal replacement

ØØ1Ø Mixed mode expression: integer constants or variables appear in an arithmetic expression with real constants or variables.

ØØ11 Invalid subscript:

a) Subscript is not an integer constant, integer variable, or legal subscript expression.

b) There are more than two subscripts (i.e., more than two dimensions.)

c) Two subscripts appear for a variable which has been defined with one dimension only.

Error Code — Description

ØØ12 Invalid constant:

a) An integer constant is not in the range of -2^{15} to 2^{15} -1.

b) A real constant is not in the approximate range of 10^{38} to 10^{-38}.

c) A constant contains an illegal character.

ØØ13 Invalid EQUIVALENCE statement:

a) Two or more of the variables appearing in an EQUIVALENCE statement are also defined in the COMMON block.

b) The variables contained in an EQUIVALENCE cause the origin of COMMON to be altered.

c) Contradictory equivalence; or equivalence between two or more arrays conflicts with a previously established equivalence.

ØØ14 Table overflow: Too many variables and statement labels appear in the program.

ØØ15 Invalid DO loop:

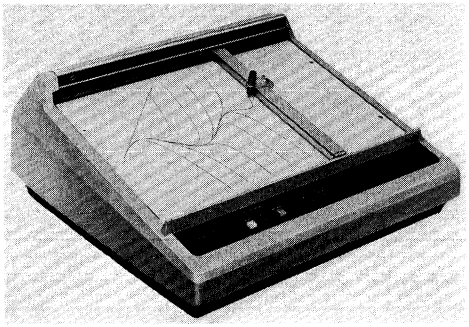
a) The terminal statement of a DO loop does not appear in the program or appears prior to the DO statement.

b) The terminal statement of a nested DO loop is not within the range of the outer DO loop.

c) DO loops are nested more than 1Ø deep.

d) Last statement in a loop is a GO TO, arithmetic IF, RETURN, STOP, PAUSE, or DO.

ØØ16 Statement function name is doubly defined.



The 7210A Graphic Plotter is easily added to your HP 2100A Computer. Up to 20 coordinate pairs per second can be accurately plotted.

BASIC Language Reference Manual

CONTENTS

COMMUNICATING WITH THE COMPUTER 1

There are many types of languages. English is a natural language used to communicate with people.

To communicate with the computer, formal languages are used. A formal language is a combination of simple English and algebra. BASIC, the Beginner's All-purpose Symbolic Instruction Code, permits the user to easily communicate with the computer. It is easy for even beginners to learn, but powerful enough for the advanced user.

Like natural languages BASIC has grammatical rules, but they are much simpler. For example, this series of BASIC statements (which calculates the average of five numbers given by you, the user) shows the fundamental rules:

```
1Ø INPUT A,B,C,D,E
2Ø LET S = (A+B+C+D+E) /5
3Ø PRINT S
4Ø GO TO 1Ø
5Ø END
```

This and the following pages show how to interpret these rules. Notice *how* the statements are written. *What* they do is explained later.

1.1 STATEMENT

This is a BASIC *statement*:

```
    1Ø INPUT A,B,C,D,E
```

Comments

A *statement* contains a maximum of 72 characters (one teletypewriter line).

A *statement* may also be called a line.

1.1.1 STATEMENT NUMBER

Each BASIC statement begins with a *statement number* (in this example, 2Ø):

```
    2Ø LET S = (A+B+C+D+E) /5
```

Comments

The number is called a *statement number* or a *line number*.

The *statement number* is chosen by you, the programmer. It may be any integer from 1 to 9999 inclusive.

Each statement has a unique *statement number*. The computer uses the numbers to keep the statements in order.

Statements may be entered in any order; they are usually numbered by fives or tens so that additional statements can be easily inserted. The computer keeps them in numerical order no matter how they are entered. For example, if statements are input in the sequence 3Ø,1Ø,2Ø; the computer arranges them in the order: 1Ø,2Ø,3Ø.

1.1.2 INSTRUCTION

The statement then gives an *instruction* to the computer (in this example, PRINT):

```
30 PRINT S
```

Comments

Instructions are sometimes called *statement types* because they identify a type of statement. For example, the statement above is a "print" statement.

1.1.3 OPERAND

If the instruction requires further details, *operands* (numeric details) are supplied (in this example, 1Ø; above, "S"):

```
4Ø GO TO 1Ø
```

Comments

The *operands* specify what the instruction act upon; for example, what is PRINTed, or where to GO.

1.1.4 FREE FORMAT IN STATEMENT

BASIC is a “free format” language—the computer ignores extra blank spaces in a statement. For example, these three statements are equivalent:

```
30 PRINT S
30 PRINT   S
30PRINTS
```

Comments

When possible, leave a space between words and numbers in a statement. This makes a program easier for people to read.

1.2 A PROGRAM

The sequence of BASIC statements given on the previous pages is called a *program*. The last statement in a program, as shown here, is an END statement.

```
1Ø INPUT A,B,C,D,E
2Ø LET S=(A+B+C+D+E) /5
3Ø PRINT S
4Ø GO TO 1Ø
5Ø END
```

Comments

The last (highest numbered) statement in a program must be an END statement.

The END statement informs the computer that the program is finished.

1.3 SPOT CHECK

Be sure you are familiar with these terms before continuing:

statement
instruction
statement type
statement number
line number
operand
program

All of these terms are defined in the context of the preceding pages.

1.4 WORKING WITH THE COMPUTER

The following pages explain how to correct mistakes and list programs.

1.4.1 ENTERING A PROGRAM

Return

The RETURN key must be pressed after each statement.

```
1Ø INPUT A,B,C,D,E,  RETURN
2Ø LET S=(A+B+C+D+E) /5  RETURN
3Ø PRINT S  RETURN
4Ø GO TO 1Ø  RETURN
5Ø END  RETURN
```

Comments

Pressing RETURN informs the computer that the statement is complete. The computer then checks the statement for mistakes.

Linefeed

The computer responds with a *linefeed* (terminal skips a line) after each statement is entered, indicating that the statement has been checked, accepted, and the computer is ready for another statement.

```
1Ø INPUT A,B,C,D,E  RETURN
linefeed
2Ø LET S = (A+B+C+D+E) /5  RETURN
linefeed
3Ø PRINT S  RETURN
linefeed
4Ø GO TO 1Ø  RETURN
linefeed
5Ø END  RETURN
linefeed
```

1.4.2 MISTAKES AND CORRECTIONS

The reverse arrow (←) key acts as a backspace, deleting the immediately preceding character.

Typing these characters:	2Ø LR←ET S=1Ø RETURN
is equivalent to typing:	2Ø LET S-1Ø RETURN
And, typing these characters:	3Ø LET←←←PRINT S RETURN
is equivalent to typing:	3Ø PRINT S RETURN

Comments

The ← character is a "shift" Ø on most terminals.

1.4.3 DELETING A STATEMENT

To delete the statement being typed, press the ESC or ALT-MODE key. This causes a \ to be printed, and deletes the entire line being typed.

```
2Ø LET S = ESC
```

NOTE: The computer responds with a \ when ESC is typed, like this:

```
2Ø LET S = \
```

To *delete* a previously typed statement, type the statement number followed by a RETURN in the following sequence:

```
5 LET S = Ø
1Ø INPUT A,B,C,D,E
2Ø LET S = (A+B+C+D+E) /5
```

To *delete* statement 5 type:

```
5  RETURN
```

1.4.4 CHANGING A STATEMENT

To *change* a previously typed statement, retype it with the desired changes. The new statement replaces the old one.

To *change* statement 5 in the above sequence, type:

```
5 LET S = 5  RETURN
```

The old statement is replaced by the new one.

Typing an ESC (or ALT-MODE) before a RETURN prevents replacement of a previously typed statement.

```
For example, typing:     5 LET ESC

                 or:     5 ESC
```

has no effect on the original statement 5.

1.5 RUNNING A PROGRAM

The program does not begin execution (does not run) until the command RUN followed by a RETURN is typed.

NOTE: The sample program (averaging 5 numbers) has been entered.

Comments	
The computer responds with a *linefeed* indicating that the command is being executed.	RUN RETURN *linefeed*
The question mark indicates that input is expected. The five numbers being averaged should be typed in, SEPARATED BY COMMAS, and followed by a RETURN.	? 95.6,87,3,5,90,82.8 RETURN *linefeed*
The answer is printed.	78.08 RETURN *linefeed*
? indicates that five more numbers are expected.	? –12.5,–50.6,–32,45.6,60 RETURN
The answer is printed.	2.1 RETURN *linefeed*
NOTE: This program continues executing indefinitely, unless terminated by the user. To terminate, type an S RETURN when more input is requested.	? S RETURN

The program is finished.

1.6 STOPPING A PROGRAM

When RUN or LIST is typed, BASIC "takes over" the terminal until the program finishes executing or the listing is complete.

To stop a program that is running or being listed, press, then release, any key.

ESC *(or any key)*

BASIC then responds with the STOP message:

```
STOP
```

Comments

Remember that: S RETURN is used to end input loops.

1.7 HOW THE PROGRAM WORKS

```
1Ø INPUT A,B,C,D,E
```

Line 1Ø tells the computer that five numbers will be input, and that they should be given the labels A,B,C,D,E in sequence. The first number input is labeled "A" by the computer, the second "B", etc. A,B,C,D, and E are called variables.

After line 1Ø is executed, the variables and their assigned values, typed in by the user, are stored. For example, using the values entered by the user in the previous example, this information is stored: A = –12.5; B = –5Ø.6; C = –32; D = 45.6; E = 6Ø.

```
2Ø LET S = (A+B+C+D+E)/5
```

Line 2Ø declares that a variable called S exists, and is assigned the value of the sum of the variables A,B,C,D,E divided by 5.

```
3Ø PRINT S
```

Line 3Ø instructs the computer to output the value of S to user's terminal.

NOTE: If the PRINT statement were not given, the value of S would be calculated and stored, but not printed. The computer must be given explicit instruction for each operation to be performed.

```
4Ø GO TO 1Ø
```

Line 4Ø tells the computer to go to line 1Ø and execute whatever instruction is there.

NOTE: A "loop" is formed by lines 1Ø to 4Ø. The sequence of statements in this loop execute until the user breaks the loop. This particular kind of loop is called an input loop (because the user must consistently input data).

TYPING: S WHEN INPUT IS REQUESTED BY A "?" IS THE ONLY WAY TO BREAK AN INPUT LOOP. Other, more controlled loops are explained later. Line 5Ø is not executed until the loop is broken by typing S when input is requested.

```
5Ø END
```

Line 5Ø informs the computer that the program is finished.

THE ESSENTIALS OF BASIC 2

This section contains enough information to allow you to use BASIC in simple applications.

Proceed at your own pace. The information in the vocabulary and operators subsections is included for completeness; experienced programmers may skip these.

The "Operators" pages contain brief descriptions, rather than explanations, of the logical operators. The novice should not expect to gain a clear understanding of logical operators from this presentation. Chapter 5 presents more details and examples of logical operations. Readers wishing to make best use of logical capabilities should consult this chapter. Those unfamiliar with logical operations should also refer to an elementary logic text.

A simple program is included at the end of this chapter for reference; it contains a running commentary on the uses of many of the BASIC statements presented in the chapter.

2.1 TERMS

2.1.1 TERM: SIMPLE VARIABLE

A letter (from A to Z); or a letter immediately followed by a digit (from Ø to 9).

```
EXAMPLES:   AØ    B
            M5    C2
            Z9    D
```

Comments

Variables are used to represent numeric values. For instance, in the statement:

```
    1Ø  LET  M5  =  96.7
```

M5 is a variable; 96.7 is the value of the variable M5.

There is one other type of variable in BASIC, the array (subscripted) variable; its use is explained in Chapter 4.

2.1.2 TERM: NUMBER

A number is defined in BASIC as a decimal number (the sign is optional) between an approximate minimum of: 1Ø$^{-38}$ (or 2^{-129}) and an approximate maximum of: 1Ø^{38} (or 2^{127}). Zero is included in this range.

EXAMPLES:	-1ØØØ8	5	3.14159	1ØE+37
	126.257	Ø	1Ø E37	1ØE–37
	16.Ø1	.Ø6784	–1Ø E37	1.ØE+2

2.1.3 TERM: E NOTATION

E notation in BASIC is a means of expressing numbers having more than six decimal digits, in the form of a decimal number raised to some power of 1Ø.

EXAMPLES: 1.ØØØØØE+Ø6 is equal to 1,ØØØ,ØØØ and is read: "1 times 1Ø to the sixth power" (1x1Ø^{6}).

1.Ø2ØØØE+Ø4 is equal to 1Ø,2ØØ

1.Ø2ØØØE–Ø4 is equal to .ØØØ1Ø2

Comments

"E" notation is used to print numbers having more than six significant digits. It may also be used for input of any number.

When entering numbers in "E" notation, leading and trailing zeros may be omitted from the number; the + sign and leading zeros may be omitted from the exponent.

The precision of numbers is 6 to 7 decimals digits (23 binary digits).

2.1.4 TERM: EXPRESSION

An expression is a combination of variables, constants and operators which evaluates to a numeric value.

EXAMPLES:

(P + 5)/27

(where P has previously been assigned a numeric value.)

Q – (N + 4)

(where Q and N have previously been assigned numeric values.)

2.1.5 TERM: ARITHMETIC EVALUATION

Arithmetic evaluation is the process of calculating the value of an expression.

2.2 OPERATORS

2.2.1 THE ASSIGNMENT OPERATOR

SYMBOL:

```
    =
```

EXAMPLES:

```
    1Ø LET A = B2 = C = Ø
    2Ø LET A9 = C5
    3Ø LET Y = (N-(R+5))/T
    4Ø LET N5 = A + B2
    5Ø LET P5 = P6=P7=A=B=98.6
```

GENERAL FORM:

LET *variable* = *expression*

Purpose

Assigns an arithmetic or logical value to a variable.

Comments

When used as an assignment operator, = is read "takes the value of," rather than "equals." It is, therefore, possible to use assignment statements such as:

```
    LET X = X+2
```

This is interpreted by BASIC as: "LET X take the value of (the present value of) X, plus two."

Several assignments may be made in the same statement, as in statements 1Ø and 5Ø above.

See Chapter 5, "Logical Operations" for a description of logical assignments.

2.2.2 RELATIONAL OPERATORS

SYMBOLS:

```
= # <> > < >= <=
```

EXAMPLES:

```
1ØØ IF A=B THEN 9ØØ
11Ø IF A+B >C THEN 91Ø
12Ø IF A+B < C+E THEN 92Ø
13Ø IF C>=D*E THEN 93Ø
14Ø IF C9<= G*H THEN 94Ø
15Ø IF P2#C9 THEN 95Ø
16Ø IF J <> K THEN 95Ø
```

Purpose

Determines the logical relationship between two expressions, as

equality:	=
inequality:	# *or* <>
greater than:	>
less than:	<
greater than or equal to:	>=
less than or equal to:	<=

Comments

NOTE: It is not necessary for the novice to understand the nature of logical evaluation of relational operators, at this point. The comments below are for the experienced programmer.

Expressions using relational operators are logically evaluated, and assigned a value of "true" or "false" (the numeric value is 1 for "true," and Ø for "false").

When the = symbol is used in such a way that it might have either an assignment or a relational function, BASIC assumes it is an assignment operator. For a description of the assignment statement using logical operators, see Chapter 5, "Logical Operations."

2.2.3 ARITHMETIC OPERATORS

SYMBOLS:

```
    ↑ * / + -
```

EXAMPLES:

```
    4Ø LET N1 = X-5
    5Ø LET C2 = N↑3
    6Ø LET A = (B-C)/4
    7Ø LET X = ((P↑2)-(Y*X))/N+Q
```

Purpose

Represents an arithmetic operation, as:

exponentiate:	↑
multiply:	*
divide:	/
add:	+
subtract:	-

Comments

The "–" symbol is also used as a sign for negative numbers. It is good practice to group arithmetic operations with parentheses when unsure of the exact order of precedence. The order of precedence (hierarchy) is:

```
↑
* /
+ –
```

with ↑ having the highest priority. Operators on the same level of priority are acted upon from left to right in a statement. See "Order of Precedence" in this Chapter for examples.

The symbols + and – are also used to indicate unary plus and unary minus. For example, negative numbers may be expressed in a statement without using parentheses:

```
10 LET A1 = –B
20 LET C2 = D ++E
30 LET B5 = B ––C
```

See "Order of Precedence" in this Chapter for examples of how unary + and unary – are interpreted.

2.2.4 THE AND OPERATOR

SYMBOL:

AND

EXAMPLES:

```
6Ø IF A9<B1 AND C#5 THEN 1ØØ
7Ø IF T7#T AND J=27 THEN 15Ø
8Ø IF P1 AND R>1 AND N AND V2 THEN 1Ø
9Ø PRINT X AND Y
```

Purpose

Forms a logical conjunction between two expressions. If both are "true," the conjunction is "true"; if one or both are "false," the conjunction is "false."

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.

Comments

The numeric value of "true" is 1, of "false" is Ø.

All non-zero values are "true." For example, statement 9Ø would print either a Ø or a 1 (the logical value of the expression X AND Y) rather than the actual numeric values of X and Y.

Control is transferred in an IF statement using AND, only when all parts of the AND conjunction are "true." For instance, example statement 8Ø requires four "true" conditions before control is transferred to statement 1Ø.

See Chapter 5, "Logical Operations" for a more complete description of logical evaluation.

2.2.5 THE OR OPERATOR

SYMBOL:

OR

EXAMPLES:

```
100 IF A>1 OR B<5 THEN 500
110 PRINT C OR D
120 LET D = X OR Y
130 IF (X AND Y) OR (P AND Q) THEN 600
```

Purpose

Forms the logical disjunction of two expressions. If either or both of the expressions are "true," the OR disjunction is "true"; if both expressions are "false," the OR disjunction is "false."

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.

Comments

The numeric values are: ‘true” = 1, “false” = Ø.

All non-zero values are “true”; all zero values are “false.”

Control is transferred in an IF statement using OR, when either or both of the two expressions evaluate to “true.”

See Chapter 5, “Logical Operations” for a more complete description of logical evaluation.

2.2.6 THE NOT OPERATOR

SYMBOL:

```
NOT
```

EXAMPLES:

```
3Ø LET X = Y = Ø
35 IF NOT A THEN 3ØØ
45 IF (NOT C) AND A THEN 4ØØ
55 LET B5 = NOT P
65 PRINT NOT (X AND Y)
7Ø IF NOT (A=B) THEN 5ØØ
```

Purpose

Logically evaluates the complement of a given expression.

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are intended for experienced programmers.

Comments

If A = Ø, then NOT A = 1; if A has a non-zero value, NOT A = Ø.

The numeric values are: “true” = 1, “false” = Ø; for example, statement 65 above would print “1”, since the expression NOT (X AND Y) is “true.”

Note that the logical specifications of an expression may be changed by evaluating the complement. In statement 35 above, if A equals zero, the evaluation would be “true” (1); since A has a numeric value of Ø, it has a logical value of “false,” making NOT A “true.”

See Chapter 5, “Logical Operations” for a more complete description of logical evaluation.

2.3 ORDER OF PRECEDENCE

The order of performing operations is:

↑	*highest precedence*
NOT unary + unary –	
* /	
+ –	
Relational Operators	
AND	
OR	*lowest precedence*

Comments

If two operators are on the same level, the order of execution is left to right, for example:

5 + 6*7	is evaluated as:	5 + (6x7)
7/14*2/5	is evaluated as:	$\frac{(7/14)x2}{5}$

Parentheses override the order of precedence in all cases, for example:

5 + (6x3)	is evaluated as:	5 + 18

and

3 + (6+(2↑2))	is evaluated as:	3 + (6+4)

Unary + and – may be used; the parentheses are assumed by BASIC. For example:

A + + B	is interpreted:	A + (+B)
C – + D –5	is interpreted:	C – (+D) –5

Leading unary + signs are omitted from output by BASIC, but remain in program listings.

2.4 STATEMENTS

Statements are instructions to the computer. They are contained in numbered lines within a program, and execute in the order of their line numbers. Statements cannot be executed without running a program. They tell the computer what to do while a program is running.

Here are some examples mentioned in Chapter 1:

LET

PRINT

INPUT

Do not attempt to memorize every detail in the "Statements" subsection; there is too much material to master in a single session. By experimenting with the sample programs and attempting to write your own programs, you will learn more quickly than by memorizing.

2.4.1 THE LET STATEMENT

EXAMPLES:

```
1Ø LET A = 5.Ø2
2Ø LET X = Y7 = Z = Ø
3Ø LET B9 = 5* (X↑2)
4Ø LET D = (3*C2↑N)/(A*(N/2))
```

GENERAL FORM:

statement number LET *variable* = *number or expression or variable variable . . .*

Purpose

Used to assign or specify the value of a variable. The value may be an expression, a number, or a variable.

Comments

The assignment statement must contain:

1. A statement number,
2. LET,
3. The variable to be assigned a value (for example, B9 in statement 3Ø above),
4. The assignment operator, an = sign,
5. The number, expression or variable to be assigned to the variable (for example, 5*(X↑2) in statement 3Ø above).

Statement 2Ø in the example shows the use of an assignment to give the same value (Ø) to several variables. This is a useful feature for initializing variables in the beginning of a program.

2.4.2 REM – COMMENTS STATEMENT

EXAMPLES:

```
    1Ø  REM–THIS IS AN EXAMPLE
    2Ø  REM:  OF REM STATEMENTS
    3Ø  REM----- /////*****!!!!!
    4Ø  REM.  STATEMENTS ARE NOT EXECUTED BY BASIC
```

GENERAL FORM:

statement number REM *any remark or series of characters*

Purpose

Allows insertion of a line of remarks or comment in the listing of a program.

Comments

Must be preceded by a line number. Any series of characters may follow REM.

REM lines are part of a BASIC program and are printed when the program is listed or punched; however, they are ignored when the program is executing.

Remarks are easier to read if REM is followed by a punctuation mark, as in the example statements.

2.4.3 INPUT STATEMENT

This program shows several variations of the INPUT statement and their effects.

Sample Program Using INPUT

```
10 INPUT  A
20 INPUT  A1,B2,C3,ZØ,Z9,E5
30 PRINT “WHAT VALUE SHOULD BE ASSIGNED TO R”;
40 INPUT R
50 PRINT A;A1;B2;C3;ZØ;Z9;E5; “R=”;R
60 GO TO 10
70 END
```

```
------------------------ RESULTS ------------------------
RUN RETURN
?1 RETURN
?2, 3, 4, 5, 6, 7  RETURN
WHAT VALUE SHOULD BE ASSIGNED TO R?27  RETURN
  1      2      3      4      5      6      7      R= 27
?1.5  RETURN
?2.5, 3.5, 4.5, 6., 7.2  RETURN
?8.1  RETURN   ? indicates that more input is expected
WHAT VALUE SHOULD BE ASSIGNED TO R?-99  RETURN
  1.5    2.5    3.5    4.5    6    7.2    8.1      R= -99
```

GENERAL FORM:

statement number INPUT *variable , variable , . . .*

Purpose

Assigns a value input from the teleprinter to a variable.

Comments

The program comes to a halt, and a question mark is printed when the INPUT statement is used. The program does not continue execution until the input requirements are satisfied.

Only one question mark is printed for each INPUT statement. The statements:

```
10 INPUT A, B2, C5, D, E, F, G
20 INPUT X
```

each cause a single "?" to be printed. The "?" generated by statement 10 requires seven input items, separated by commas, while the "?" generated by statement 20 requires only a single input item

The only way to terminate or exit a program when input is required is entering: S RETURN. Note that the S ends the program; it must be restarted with the RUN command.

Relevant Diagnostics:

? indicates that input is required.

NOTE: A ? is printed on the terminal when more numbers are required to satisfy an input statement (usually, too few numbers were typed). The ? continues to be printed after each response until enough numbers are typed in.

See PRINT in this chapter for output variations.

2.4.4 PRINT STATEMENT

This sample program gives a variety of examples of the PRINT statement. The results are shown below.

```
 1Ø LET  A=B=C=1Ø
 2Ø LET  D1=E9=2Ø
 3Ø PRINT  A,B,C,D1,E9
 4Ø PRINT  A/B,B/C/D1+E9
 5Ø PRINT "NOTE THE POWER TO EVALUATE AN
      EXPRESSION AND PRINT THE"
 6Ø PRINT "VALUE IN THE SAME STATEMENT."
 7Ø PRINT
 8Ø PRINT
 9Ø REM* "PRINT" WITH NO OPERAND CAUSES THE
      TELEPRINTER TO SKIP A LINE.
1ØØ PRINT " 'A'  DIVIDED BY 'E9'  ="; A/E9
11Ø PRINT  "11111", "22222", "33333", "44444",
      "55555", "66666"
12Ø PRINT  "11111"; "22222"; "33333"; "44444";
      "55555"; "66666"
13Ø END
```

------------------------RESULTS------------------------

```
RUN  RETURN

  1Ø        1Ø        1Ø        2Ø        2Ø
   1        2Ø.Ø5

NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT
THE VALUE IN THE SAME STATEMENT.

'A' DIVIDED BY 'E9'  =  .5
11111       22222       33333       44444       55555       66666
111112222233333444445555566666
```

NOTE: The "," and ";" used in statements 11Ø and 12Ø have very different effects on the format.

GENERAL FORM:

statement number PRINT *expression , expression , . . .*

or

statement number PRINT *"any text" ; expression ; . . .*

or

statement number PRINT *"text" ; expression ; "text", "text" , . . .*

or

statement number PRINT *any combination of text and/or expressions*

or

statement number PRINT

Purpose

Causes the *expression* or *"text"* to be output to the terminal.

Causes the teleprinter to skip a line when used without an operand.

Comments

Note the effects of , and ; on the output of the sample program. If a comma is used to separate PRINT operands, five fields are printed per teleprinter line. If semicolon is used, up to twelve "packed" numeric fields are output per teleprinter line (72 characters).

Text in quotes is printed literally.

NOTE: A variable name is considered as a simple expression by BASIC. For example, a statement for the first general form shown above might be:

```
100 PRINT A1, B2, C3
```

or

```
100 PRINT A, Z, X, T9
```

where the variables represent numeric expressions.

Remember that variable values must be defined in an assignment, INPUT, READ or FOR statement before being used in a PRINT statement.

Although the format of the PRINT statement is "automatic" to help beginning programmers, the experienced programmer may use several features to control his output format.

Each line output to the terminal is divided into five print fields when commas are used as separators (as in statement 30 in the sample program). The fields begin at print spaces 0, 15, 30, 45, and 60. The first four fields contain fifteen spaces, and the last field contains twelve. The comma signals the computer to move to the next print field, or if in the last field, to move to the next line.

More information may be printed on a line by using semicolons as separators. Twelve numbers may be printed per line by using semicolons. (See the output from statements 110 and 120 in the sample program for an example of the differences in the two separators.)

Spacing within a print field depends on the value and type of the number being printed. A number is always printed in a field larger than itself and is left-justified. The space required for a number is determined by these formulas:

Value of Number	Type of Number	Output Field Size
$-999 \leqslant n \leqslant +999$	Integer	$\Lambda^{xxx}\Lambda\Lambda$*
$-32768 \leqslant n \leqslant -1ØØØ$ $+1ØØØ \leqslant n \leqslant +32767$	Integer	$\Lambda^{xxxxx}\Lambda\Lambda$
$.1 \leqslant n \leqslant 999999.5$	Large Integer or Real	$\Lambda^{xxxxxxxx}\Lambda\Lambda\Lambda\Lambda$ (Decimal point printed as one of the x's; trailing zeros suppressed.)
$n < .1$ $999999.5 < n$	Large Integer or Real	$\Lambda^{x.xxxxxE\pm ee}\Lambda\Lambda\Lambda$

*The Λ symbol indicates a space.

Ending a PRINT statement with a semicolon causes the output to be printed on the same line, rather than generating a RETURN *linefeed* after the statement is executed. For example, the sequence:

```
    2Ø LET X = 1
    3Ø PRINT X;
    4Ø LET X=X+1
    5Ø GO TO 3Ø
     .
     .
     .
```

produces output in this format:

```
    1   2   3   4   5   6   7   8   9   1Ø  11  12
    13  14  15  16  17  18  19  2Ø  21  22  23  24
```

Similarly, ending a PRINT statement with a comma causes output to fill all five fields on a line before moving to the next line. The trailing comma in statement 3Ø in the sequence:

```
2Ø LET X = 1
3Ø PRINT X,
4Ø LET X=X+1
5Ø GO TO 3Ø
 .
 .
 .
```

produces output in this format:

```
1       2       3       4       5
6       7       8       9       1Ø
11      12      13      14      15
```

A PRINT statement without an operand (statements 7Ø and 8Ø in the sample program) generates a RETURN *linefeed.*

Three general rules for planning output formats are:

1. If a number is an integer with a value between –32768 and +32767, inclusive, the decimal point is not printed.

2. If the number is an integer out of the above range or if the number is real and has an absolute value between .1 and 999999.5, the number is rounded to six digits and printed with a decimal point. Zeros trailing the decimal point are suppressed.

3. If a number is either greater than 999999.5 or less than .1, it is rounded to six places; the teletypewriter then prints a space (if positive) or minus sign (if negative), the first digit, the decimal point, the next five digits, the letter E (indicating exponent), the sign of the exponent, and the exponent.

Unlike numbers, strings (characters enclosed in quotation marks) are printed without leading or trailing blanks when the semicolon separator is used. For example, the program:

```
15 PRINT “ANTIDISESTABLISH”;
2Ø PRINT “MENTARIANISM”
3Ø END
```

when executed prints the two strings adjacent to one another:

```
RUN
ANTIDISESTABLISHMENTARIANISM
```

See the description of the TAB function in Chapter 3 for more information on controlling output format.

2.4.5 GO TO STATEMENT

EXAMPLES:

```
    1Ø  LET  X  =  2Ø
     .
     .
     .
    5Ø  GOTO  1ØØ
    8Ø  GOTO  1Ø
```

GENERAL FORM:

statement number GO TO *statement number*

Purpose

Transfers control to the specified statement.

Comments

GO TO may be written: GOTO or GO TO.

This statement must be followed by the statement number to which control is transferred.

GO TO overrides the normal execution sequence of statements in a program.

Useful for repeating a task infinitely, or "jumping" (GOing TO) another part of a program if certain conditions are present.

GO TO should not be used to enter FOR-NEXT loops; doing so may produce unpredictable results or fatal errors. (See "FOR . . . NEXT" in this section for details on loops.)

To get out of a GO TO loop, press any key.

2.4.6 IF . . . THEN

SAMPLE PROGRAM:

```
10 LET N = 10
2Ø READ X
3Ø IF X <=N THEN 6Ø
4Ø PRINT "X IS OVER"; N
5Ø GO TO 1ØØ
6Ø PRINT "X IS LESS THAN OR EQUAL TO"; N
7Ø GO TO 2Ø
8Ø STOP
 .
 .
 .
```

GENERAL FORM:

statement number IF *expression* | *relational op* | *expression*
THEN *statement number*

Purpose

Transfers control to a specified statement if a specified condition is true.

Comments

Sometimes described as a conditional transfer; "GO TO" is implied by IF . . . THEN, if the condition is true. In the example above, if X<=1Ø, the message in statement 6Ø is printed (statement 6Ø is executed).

Since numbers are not always represented exactly in the computer, the = operator should be used carefully in IF . . . THEN statements. Limits, such as <=,>=, etc. should be used in an IF expression, rather than =, whenever possible.

If the specified condition for transfer is not true, the program will continue executing in sequence. In the example above, if X>1Ø, the message in statement 4Ø prints.

The relational operator is optional in logical evaluations.

See Chapter 5, "Logical Operations," for a more complete description of logical evaluation.

2.4.7 FOR . . . NEXT

EXAMPLES:

```
100 FOR P1 = 1 TO 5
110 FOR Q1 = N TO X
120 FOR R2 = N TO X STEP 2.5
130 FOR S  = 1 TO X STEP Y
140 NEXT S
150 NEXT R2
160 NEXT Q1
170 NEXT P1
```

Sample Program – Variable Number of Loops

```
 40 PRINT “HOW MANY TIMES DO YOU WANT TO LOOP”;
 50 INPUT A
 60 FOR J = 1 TO A
 70 PRINT “THIS IS LOOP”; J
 80 READ N1, N2, N3
 90 PRINT “THESE DATA ITEMS WERE READ:” N1; N2; N3
100 PRINT “SUM =”; (N1+N2+N3)
110 NEXT J
120 DATA 5, 6, 7, 8, 9, 10, 11, 12
130 DATA 13, 14, 15, 16, 17, 18, 19, 20, 21
140 DATA 22, 23, 24, 25, 26, 27, 28, 29, 30
150 DATA 31, 32, 33, 34
160 END
```

GENERAL FORM:

statement number FOR *simple variable = initial value* TO *final value*

or

statement no. FOR *simple var. = initial value* TO *final value* STEP *step value*

statement number NEXT *simple variable*

NOTE: The same simple variable must be used in both the FOR and NEXT statements of a loop.

Purpose

Allows controlled repetition of a group of statements within a program.

Comments

Initial value, final value and *step value* may be any expression.

STEP and *step value* are optional; if no *step value* is specified, the computer will automatically increment by one each time it executes the loop.

How the loop works:

The simple variable is assigned the value of the *initial value*; the value of the simple variable is increased by 1 (or by the *step value*) each time the loop executes. When the value of the *simple variable* passes the *final value*, control is transferred to the statement following the "NEXT" statement.

The *initial*, *final*, and *step values* are all evaluated upon entry to the loop and remain unchanged after entry. For example,

```
FOR I = 1 TO I + 5
```

goes from 1 to 6; that is, the *final value* does not "move" as I increases with each pass through the loop.

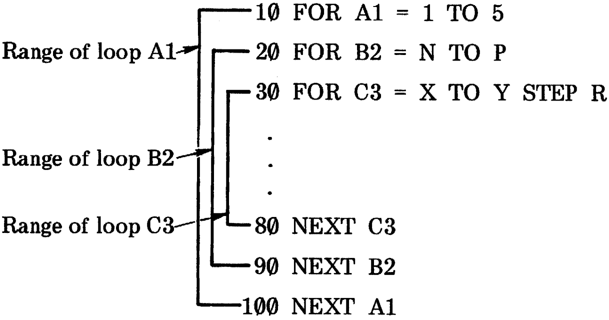
For further details on the STEP feature, see "FOR . . . NEXT with STEP" in Chapter 3.

Try running the sample program if you are not sure what happens when FOR . . . NEXT loops are used in a program.

2.4.8 NESTING FOR . . . NEXT LOOPS

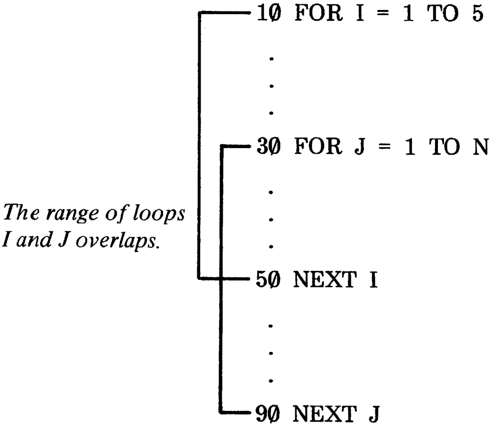
Several FOR . . . NEXT loops may be used in the same program; they may also be nested (placed inside one another). There are two important features of FOR . . . NEXT loops:

1. FOR . . . NEXT loops may be nested.



```
10 FOR A1 = 1 TO 5
20 FOR B2 = N TO P
30 FOR C3 = X TO Y STEP R
 .
 .
 .
80 NEXT C3
90 NEXT B2
100 NEXT A1
```

2. The range of FOR . . . NEXT loops may not overlap. The loops in the example above are nested correctly. This example shows improper nesting.



```
10 FOR I = 1 TO 5
 .
 .
 .
30 FOR J = 1 TO N
 .
 .
 .
50 NEXT I
 .
 .
 .
90 NEXT J
```

The range of loops I and J overlaps.

2.4.9 READ, DATA AND RESTORE STATEMENTS

Sample Program using READ and DATA

```
15 FOR I=1 TO 5
20 READ A
40 LET X=A↑2
45 PRINT A;” SQUARED =”;X
50 NEXT I
55 DATA 5.24,6.75,30.8,72.65,89.72
60 END
```

Each data item may be read only once in this program. BASIC keeps track of data with a “pointer.” When the first READ statement is encountered, the “pointer” indicates that the first item in the first DATA statement (the one with the lowest statement number) is to be read; the pointer is then moved to the second item of data, and so on.

In this example, after the loop has executed five times, the pointer remains at the end of the data list. To reread the data, it is necessary to reset the pointer. A RESTORE statement moves the pointer back to the first data item.

Sample Program Using RESTORE with READ and DATA.

```
20 FOR I = 1 TO 5
30 READ A
40 LET X=A↑2
50 PRINT A; “SQUARED =”;X
60 NEXT I
80 RESTORE
```

```
100 FOR J=1 TO 5
110 READ B
120 LET Y=B↑4
130 PRINT B; "TO THE FOURTH POWER =";Y
140 NEXT J
150 DATA 5.24,6.75,30.8,72.65,89.72
160 END
```

GENERAL FORM:

statement number READ *variable* , *variable* , . . .

statement number DATA *number* , *number* , . . .

statement number RESTORE

Purpose

The READ statement instructs BASIC to read an item from a DATA statement.

The DATA statement is used for specifying data in a program. The data is read in sequence from first to last DATA statements, and from left to right within the DATA statement.

The RESTORE statement resets the pointer to the first data item, allowing data to be reread.

2.5 PROGRAM HALTS — TEMPORARY, PERMANENT

2.5.1 WAIT STATEMENT

EXAMPLE:

```
900 WAIT (1000)
990 WAIT (3000)
```

GENERAL FORM:

statement number WAIT *(expression | max. value of 32767)*

Purpose

Introduces delays into a program. WAIT causes the program to wait the specified number of milliseconds (maximum 32767 milliseconds) before continuing execution.

Comments

The time delay produced by WAIT is not precisely the number of milliseconds specified because there is no provision to account for time elapsed during calculation or terminal-computer communication.

One millisecond = 1/1ØØØ second.

2.5.2 END AND STOP STATEMENTS

EXAMPLES:

```
    2ØØ IF A # 27.5 THEN 35Ø
      .
      .
      .
    3ØØ STOP
      .
      .
      .
    5ØØ IF B # A THEN 9999
      .
      .
      .
    55Ø PRINT "B = A"
    6ØØ END
    9999 END
```

GENERAL FORM:

any statement number STOP

any statement number END

highest statement number in program END

Purpose

Terminates execution of the program.

Comments

The highest numbered statement in the program must be an END statement.

END and STOP statements *may* be used in any portion of the program to terminate execution.

END and STOP have identical effects; the only difference is that the highest numbered statement in a program must be an END statement.

The RUN command is used to rerun programs terminated by STOP or END statements; execution always begins at the lowest numbered statement in the program.

2.6 SAMPLE PROGRAM

If you understand the effects of the statement types presented up to this point, skip to the "COMMANDS" section.

The sample program on the next two pages uses several BASIC statement types.

Running the program gives a good idea of the various effects of the PRINT statement on teleprinter output. If you choose to run the program, you may save time by omitting the REM statements.

After running the program, compare your output with that shown under "RUNNING THE SAMPLE PROGRAM." If there is a difference, LIST your version and compare it with the one presented on the next two pages. Check the commas and semicolons; they must be used carefully.

2.6.1 LISTING OF SAMPLE PROGRAM

```
 10 REMARK: "REMARK" OR "REM" IS USED TO INDICATE
    REMARKS OR COMMENTS
 20 REMARK: THE USER WANTS TO INCLUDE IN THE TEXT OF
    HIS PROGRAM.
 30 REM: THE COMPUTER LISTS AND PUNCHES THE "REM"
    LINE, BUT DOES NOT
 40 REM: EXECUTE IT.
 50 REM: "PRINT" USED ALONE GENERATES A "RETURN"
    "LINEFEED"
 60 PRINT
 70 PRINT "THIS PROGRAM WILL AVERAGE ANY GROUP OF
    NUMBERS YOU SPECIFY."
 80 PRINT
 90 PRINT "IT WILL ASK ALL NECESSARY QUESTIONS AND
    GIVE INSTRUCTIONS."
100 PRINT
110 PRINT "PRESS THE RETURN KEY AFTER YOU TYPE YOUR
    REPLY."
120 PRINT
130 PRINT
140 REM: FIRST, ALL VARIABLES USED IN THE PROGRAM ARE
    INITIALIZED
150 REM: TO ZERO (THEIR VALUE IS SET AT ZERO).
160 LET A=N=R1=S=0
180 REM: NOW THE USER WILL BE GIVEN A CHANCE TO
    SPECIFY HOW MANY
190 REM: NUMBERS HE WANTS TO AVERAGE.
200 PRINT "HOW MANY NUMBERS DO YOU WANT TO AVERAGE";
210 INPUT N
```

```
220 PRINT
230 PRINT “O.K., TYPE IN ONE OF THE “;N;” NUMBERS AFTER
    EACH QUES. MARK.”
240 PRINT “DON’T FORGET TO PRESS THE RETURN KEY AFTER
    EACH NUMBER.”
250 PRINT
260 PRINT “NOW, LET’S BEGIN”
270 PRINT
280 PRINT
300 REM: “N” IS NOW USED TO SET UP A “FOR-NEXT” LOOP
    WHICH WILL READ
310 REM: 1 TO “N” NUMBERS AND KEEP A RUNNING TOTAL.
320 FOR I=1 TO N
330 INPUT A
340 LET S=S+A
350 NEXT I
360 REM: “I” IS A VARIABLE USED AS A COUNTER FOR THE
    NUMBER OF TIMES
370 REM: THE TASK SPECIFIED IN THE “FOR-NEXT” LOOP IS
    PERFORMED.
380 REM: “I” INCREASES BY 1 EACH TIME THE LOOP IS
    EXECUTED.
390 REM: “A” IS THE VARIABLE USED TO REPRESENT THE
    NUMBER TO BE
400 AVERAGED. THE VALUE OF “A” CHANGES EACH TIME
    THE
410 REM: USER INPUTS A NUMBER.
420 REM: “S” WAS CHOSEN AS THE VARIABLE TO REPRESENT
    THE SUM
430 REM: OF ALL NUMBERS TO BE AVERAGED.
440 REM: AFTER THE LOOP IS EXECUTED “N” TIMES, THE
    PROGRAM CONTINUES.
```

```
46Ø REM:  A SUMMARY IS PRINTED FOR THE USER.
47Ø PRINT
48Ø PRINT
49Ø PRINT N; “NUMBERS WERE INPUT.”
5ØØ PRINT
51Ø PRINT “THEIR SUM IS:”;S
52Ø PRINT
53Ø PRINT “THEIR AVERAGE IS:”;S/N
54Ø PRINT
55Ø PRINT
57Ø REM: NOW THE USER WILL BE GIVEN THE OPTION OF
     QUITTING OR
58Ø REM:  RESTARTING THE PROGRAM.
59Ø PRINT “DO YOU WANT TO AVERAGE ANOTHER GROUP OF
     NUMBERS?”
6ØØ PRINT
61Ø PRINT “TYPE 1 IF YES, Ø IF NO”
62Ø PRINT “BE SURE TO PRESS THE RETURN KEY AFTER YOUR
     ANSWER.”
63Ø PRINT
64Ø PRINT “YOUR REPLY”;
65Ø INPUT R1
66Ø IF R1=1 THEN 12Ø
67Ø REM: THE FOLLOWING LINES ANTICIPATE A MISTAKE IN
     THE REPLY.
68Ø IF R1#Ø THEN 7ØØ
69Ø GO TO 72Ø
7ØØ PRINT “TO REITERATE, YOU SHOULD TYPE 1 IF YES, Ø
     IF NO.”
71Ø GO TO 64Ø
72Ø END
```

2.6.2 RUNNING THE SAMPLE PROGRAM

```
RUN  RETURN

THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU
     SPECIFY.

IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUC-
     TIONS.

PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY.

HOW MANY NUMBERS DO YOU WANT TO AVERAGE? 5  RETURN

O.K., TYPE IN ONE OF THE 5 NUMBERS AFTER EACH QUES.
     MARK.

DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH
     NUMBER.

NOW, LET'S BEGIN

? 99  RETURN

? 87.6  RETURN

? 92.7  RETURN

? 79.5  RETURN

? 84  RETURN

5  NUMBERS WERE INPUT.

THEIR SUM IS:  442.8

THEIR AVERAGE IS:  88.56

DO YOU WISH TO AVERAGE ANOTHER GROUP OF NUMBERS?

TYPE 1 IF YES, Ø IF NO

BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.

YOUR REPLY?  2  RETURN

TO REITERATE, YOU SHOULD TYPE 1 IF YES, Ø IF NO.

YOUR REPLY?  1  RETURN

HOW MANY NUMBERS DO YOU WISH TO AVERAGE?  S  RETURN
```

2.7 COMMANDS

Note the difference between commands and statements. (See "Statements" in this section.)

Commands are also instructions. They are executed immediately, do not have line numbers, and may not be used in a program. They are used to manipulate programs, and for utility purposes.

Do not try to memorize all of the details in the COMMANDS subsection. The various commands and their functions will become clear to you as you begin to write your own programs.

2.7.1 RUN

EXAMPLE: RUN RETURN

GENERAL FORM: RUN

Purpose

Starts execution of a program at the lowest numbered statement.

Comments

A running program may be terminated by pressing any key. To terminate a running program at some point when input is required, type:

```
S RETURN
```

2.7.2 LIST

EXAMPLE:

```
LIST RETURN
    or
LIST 1ØØ RETURN
```

GENERAL FORM:

LIST

LIST *statement number*

Purpose

Produces a listing of all statements in a program (in statement number sequence) when no statement number is specified.

When a statement number is specified, the listing begins at that statement.

Comments

A listing may be stopped by pressing any key.

2.7.3 SCRATCH

EXAMPLE: SCRATCH RETURN

GENERAL FORM: SCRATCH

or

SCR

Purpose

Deletes (from memory) the program currently being accessed from the teleprinter.

Comments

SCRATCH erases *everything* in the user's area of computer memory.

SCRATCHed programs are not recoverable. For information about saving programs on paper tape, see the PLIST command in this section.

2.7.4 TAPE

EXAMPLE: TAPE RETURN

GENERAL FORM: TAPE

or

TAP

Purpose

Informs the computer that following input is from paper tape being read from the terminal tape reader.

Comments

BASIC responds to the TAPE command with a *linefeed.*

TAPE suppresses *linefeeds* following statements.

Error messages are printed as the tape is input; the tape reader is held inactive while they are being printed.

2.7.5 PTAPE

EXAMPLE: PTAPE RETURN

GENERAL FORM:

```
PTAPE
  or
PTA
```

Purpose

Causes the computer to read in a program from the punched tape photoreader.

Comments

If the computer does not have a photoreader, the message:

```
STOP
READY
```

is printed on the terminal, and BASIC waits for further input.

BASIC responds to the PTAPE command with a *linefeed.*

2.7.6 PLIST

EXAMPLE: PLIST RETURN

GENERAL FORM: PLIST

or

PLIST *statement number*

Purpose

Causes the program in memory to be punched onto paper tape, with leading and trailing guide holes; also produces a listing of the program on the HP modified ASR-33 terminal; one listing is produced on the HP modified ASR-35 in 'KT' mode.

Comments

Be sure to press the "ON" button on the terminal paper tape punch before pressing *return* after PLIST.

If there is no paper tape punch on the terminal, a listing is printed.

BASIC uses the high-speed punch if available, otherwise the terminal punch is used.

ADVANCED BASIC 3

This section describes further capabilities of BASIC.

The experienced programmer has the option of skipping the "Vocabulary" subsection, and briefly reviewing the commands and functions presented here. Matrices are explained in the next chapter.

The inexperienced programmer need not spend a great deal of time on programmer-defined and standard functions. They are shortcuts, and some programming experience is necessary before their applications become apparent.

3.1 TERMS

3.1.1 TERM: ROUTINE

A sequence of program statements which produces a certain result.

Purpose

Routines are used for frequently performed operations, saving the programmer the work of defining an operation each time he uses it, and saving computer memory space.

Comments

A routine may also be called a program, subroutine, or sub-program.

The task performed by a routine is defined by the programmer.

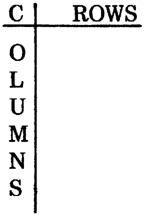
Examples of routines and subroutines are given in this section.

3.1.2 TERM: ARRAY OR MATRIX

An ordered collection of numeric data (numbers).

Comments

Arrays are divided into columns (vertical) and rows (horizontal):



Arrays may have one or two dimensions. For example,

1.Ø
2.1
3.2
4.3   is a one-dimensional array,

while,   6 , 5 , 4
3 , 2 , 1
Ø , 9 , 8   is a two-dimensional array.

Array elements are referenced by their row and column position. For instance, if the two examples above were arrays A and Z respectively, 2.1 would be A(2); similarly, Ø would be Z(3,1). The references to array elements are called subscripts, and set apart with parentheses. For example, P(1,5) references the fifth element of the first row of array P; 1 and 5 are subscripts. In X(M,N) M and N are the subscripts.

3.1.3 TERM: STRING

Zero to 65 teleprinter characters enclosed by quotation marks (one line on a teleprinter terminal).

Comments

Sample strings: "ANY CHARACTERS!?*/---"

"TEXT 1234567 . . ."

Quotation marks may not be used within a string. Strings are used only in PRINT statements.

The statement number, PRINT, and quotation marks are not included in the 65 character count. Each statement may contain up to 72 characters. Maximum string length is 72 characters minus 6 characters for "PRINT", two for the quotation marks, and the number of characters in the statement number.

3.1.4 TERM: FUNCTION

The mathematical relationship between two variables (X and Y, for example) such that for each value of X there is one and only one value of Y.

Comments

The independent variable in a function is called an argument; the dependent variable is the function value. For instance, if X is the argument, the function value is the square root of X, and Y takes the value of the function.

3.1.5 TERM: WORD

The amount of computer memory space occupied by two teleprinter characters.

Comments

Numbers require two words of memory space when stored as numbers. When used within a string, numbers require 1/2 word of space per character in the number.

3.2 SUBROUTINES AND FUNCTIONS

The following pages explain BASIC features useful for repetitive operations — subroutines, programmer-defined functions and standard functions.

The programmer-defined features, such as GOSUB, FOR . . . NEXT with STEP, and DEF FN become more useful as the user gains experience and learns to use them as shortcuts.

Standard mathematical and trigonometric functions are convenient timesavers for programmers at any level. They are treated as numeric expressions by BASIC.

3.2.1 GOSUB . . . RETURN

EXAMPLE:

```
    5Ø READ A2
    6Ø IF A2<1ØØ THEN 8Ø
    7Ø GOSUB 4ØØ
     .
     .
     .
    38Ø STOP     (STOP, END, or GO TO frequently precedes the
                 first statement of a subroutine to prevent accidental
                 entry.)
    39Ø REM- -THIS SUBROUTINE ASKS FOR A 1 OR Ø REPLY.
    4ØØ PRINT "A2 IS>1ØØ"
```

```
410 PRINT "DO YOU WANT TO CONTINUE";
420 INPUT N
430 IF N #Ø THEN 45Ø
44Ø LET A2 = Ø
45Ø RETURN
 .
 .
 .
6ØØ END
```

GENERAL FORM:

statement number GOSUB statement number starting subroutine

.

.

.

statement number RETURN

Purpose

GOSUB transfers control to the specified statement number.

RETURN transfers control to the statement following the GOSUB statement which transferred control.

GOSUB . . . RETURN eliminates the need to repeat frequently used groups of statements in a program.

Comments

The portion of the program to which control is transferred must logically end with a RETURN statement. RETURN statements may be used at any desired exit point in a subroutine.

Subroutines should be entered only with GOSUB statements rather than GO TO's, to avoid unexpected RETURN errors (which cause the program to stop execution).

GOSUB . . . RETURN's may be logically "nested" to a level of nine during execution. There is no limit on physical nesting in a program.

This sequence shows logically nested GOSUB's:

```
 1Ø INPUT
 2Ø GOSUB 1ØØ
  .
  .
  .
1ØØ IF C>Ø THEN 12Ø
11Ø LET C=-C
12Ø GOSUB 2ØØ
13Ø RETURN
  .
  .
  .
2ØØ LET A=SQR(C)
21Ø LET C=SQR(A)
22Ø RETURN
3ØØ END
```

The order in which this program is executed is:

when C>0:

10
20
100
120
200
210
220
130
statements after 20

when C<=0:

10
20
100
110
120
200
210
220
130
statements after 20

Note that the first GOSUB executed is 100, and that the second GOSUB (200) is "nested" in the first, that is, the second GOSUB is executed before the RETURN in the first GOSUB. The structure is simple:

```
GOSUB #1
  .
  .
  .
GOSUB #2
  .
  .
  .
RETURN FOR #2
  .
  .
  .
RETURN FOR #1
```

GOSUB #2 is logically nested inside GOSUB #1; a maximum of 9 GOSUB's may be nested in this manner.

3.2.2 FOR . . . NEXT WITH STEP

EXAMPLES:

```
    2Ø FOR I5 = 1 TO 2Ø STEP 2
    4Ø FOR N2 = Ø TO -1Ø STEP -2
    8Ø FOR P = 1 TO N STEP X5
    9Ø FOR X = N TO W STEP (N↑2-V)
     .
     .
     .
```

GENERAL FORM:

statement no. FOR simple var. = expression TO expression STEP expression

Purpose

Allows the user to specify the size of the increment of the FOR variable.

Comments

The step size need not be an integer. For instance,

```
    1ØØ FOR N = 1 TO 2 STEP .Ø1
```

is a valid statement which produces approximately 1ØØ loop executions, incrementing N by .Ø1 each time.

Since no binary computer represents all decimal numbers exactly, round-off errors may increase or decrease the number of steps when a non-integer step size is used.

A step size of 1 is assumed if STEP is omitted from a FOR statement.

A negative step size may be used, as shown in statement 4Ø above.

3.2.3 GENERAL MATHEMATICAL FUNCTIONS

EXAMPLES:

```
642 PRINT EXP(N); ABS(N)
652 IF RND (Ø)>=.5 THEN 9ØØ
662 IF INT (R) # 5 THEN 91Ø
672 PRINT SQR (X); LOG (X)
```

GENERAL FORM:

The general mathematical functions may be used as expressions, or as parts of an expression.

Purpose

Facilitates the use of common mathematical functions by pre-defining them as:

ABS *(expression)* — the absolute value of the expression;

EXP *(expression)* — the constant e raised to the power of the expression value (in statement 642 above, e↑N);

INT *(expression)* — the largest integer $\leqslant$ the expression;

LOG *(expression)* — the logarithm of the positively valued expression to the base e;

RND *(expression)* a random number between 1 and Ø; the expression is a dummy argument;

SQR *(expression)* the square root of the positively valued expression.

Comments

The RND function is restartable; the sequence of random numbers using RND is identical each time a program is RUN.

3.2.4 TRIGONOMETRIC FUNCTIONS

EXAMPLES:

```
    5ØØ PRINT SIN(X): COS(Y)
    51Ø PRINT 3*SIN(B); TAN (C2)
    52Ø PRINT ATN (22.3)
    53Ø IF SIN (A2) <1 THEN 8ØØ
    54Ø IF SIN (B3) = 1 AND SIN(X) <1 THEN 9Ø
```

Purpose

Facilitates the use of common trigonometric functions by pre-defining them, as:

SIN *(expression)* the sine of the expression (in radians);

COS *(expression)* the cosine of the expression (in radians);

TAN *(expression)* the tangent of the expression (in radians);

ATN *(expression)* the arctangent of the expression (returns the angle in radians.)

Comments

The function is of the value of the expression (the value in parentheses, also called the argument).

The trigonometric functions may be used as expressions or parts of an expression.

3.2.5 DEF FN — FUNCTION DEFINITION

EXAMPLE:

```
6Ø DEF FNA (B2) = A↑2 + (B2/C)
7Ø DEF FNB (B3) = 7*B3↑2
8Ø DEF FNZ (X) = X/5
```

GENERAL FORM:

statement no. DEF FN single letter A to Z
(simple var.) = expression

Purpose

Allows the programmer to define functions.

Comments

The simple variable is a “dummy” variable whose purpose is to indicate where the actual argument of the function is used in the defining expression. After a function has been defined, the value of that function is referenced whenever the function is used by the programmer. For example, in this sequence:

```
1Ø LET Y = 1ØØ
2Ø DEF FNA (Y) = Y/1Ø
3Ø PRINT FNA (Y)
4Ø END
RUN
1Ø
```

When FNA (Y) is called for in statement 3Ø, the formula defined for FNA in statement 2Ø is used to determine the value printed.

A maximum of 26 programmer-defined functions are possible in a program (FNA to FNZ).

Any operand in the program may be used in the defining expression; however such circular definitions as:

```
1Ø DEF FNA (Y) = FNB (X)
2Ø DEF FNB (X) = FNA (Y)
```

cause infinite looping.

See the vocabulary at the beginning of this section for a definition of “function” and an explanation of “arguments.”

3.2.6 COM STATEMENT

EXAMPLES:

1 COM A(1Ø), B(3,3) *first program*

1 COM C(5), D(5), F(3,3) *subsequent program*

GENERAL FORM:

lowest statement no. COM subscripted array var., separated by commas

Purpose

Allows a BASIC program to store data in memory for retrieval by a subsequent BASIC program.

Comments

The data designated by a COM statement is accessible only as an array; since COM designates a common array of data, the same array variable cannot appear in both DIM and COM statements within a program.

COM must be the first statement entered and the lowest numbered statement in a program.

The common area is a block of contiguous data in memory (two computer words per number). The storage space is allotted in the order that the arrays appear in the COM statement; the elements within an array are stored row by row.

It is the user's responsibility to see that the portions of the common area are accessed properly by subsequent programs. For example, if the first program starts with the statement 1 COM A(1Ø), B(3,3) and a subsequent program with 1 COM C(5), D(5), F(3,3), the common storage area elements are assigned as follows:

Element Position	First Program Reference	Second Program Reference
1	A(1)	C(1)
2	A(2)	C(2)
3	A(3)	C(3)
4	A(4)	C(4)
5	A(5)	C(5)
6	A(6)	D(1)
7	A(7)	D(2)
8	A(8)	D(3)
9	A(9)	D(4)
1Ø	A(1Ø)	D(5)
11	B(1,1)	F(1,1)
12	B(1,2)	F(1,2)
13	B(1,3)	F(1,3)
14	B(2,1)	F(2,1)
15	B(2,2)	F(2,2)
16	B(2,3)	F(2,3)
17	B(3,1)	F(3,1)
18	B(3,2)	F(3,2)
19	B(3,3)	F(3,3)

A reference in the first program to B(1,1) accesses the same element as a reference to F(1,1) in the second program. If A contained only 9 elements, however, the B(1,1) and F(1,1) references would access different elements.

The length of the common area may vary between programs, but for any two programs, information may be transferred only via the portion which is common to both.

If the first program declares 1 COM A(10), B(5,5) and a succeeding program contains 1 COM D(10), E(5,5), F(10), the values of F would be unpredictable. If the second program contained 1 COM D(10) only, the contents of B would be destroyed.

3.2.7 THE TAB AND SGN FUNCTIONS

EXAMPLES:

```
500 IF SGN (X) # 0 THEN 800
510 LET Y = SGN(X)
520 PRINT TAB (5); A2; TAB (20)"TEXT"
530 PRINT TAB (N),X,Y,Z2
540 PRINT TAB (X+2) "HEADING"; R5
```

GENERAL FORM:

The TAB and SGN may be used as expressions, or parts of an expression. The function forms are:

TAB (*expression indicating number of spaces to be moved*)

SGN (*expression*)

Purpose

TAB *(expression)* is used only in a PRINT statement, and causes the terminal typeface to move to the space number specified by the expression (0 to 71). The *expression* value after TAB is rounded to the nearest integer. Expression values greater than 71 cause a RETURN *linefeed* to be generated.

SGN *(expression)* returns a 1 if the expression is greater than 0, returns a 0 if the expression equals 0, returns a –1 if the expression is less than 0.

MATRICES 4

4.1 TERMS

This section explains matrix manipulation. It is intended to show the matrix capabilities of BASIC and assumes that the programmer has some knowledge of matrix theory.

4.1.1 TERM: MATRIX (ARRAY)

An ordered collection of numeric data (numbers).

Matrix elements are referenced by subscripts following the matrix variable, indicating the row and column of the element. For example, if matrix A is:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

the element 5 is referenced by A(2,2); likewise, 8 is A(3,2).

See 3.1.2 for a more complete description of matrices.

4.1.2 DIM STATEMENT

EXAMPLES:

```
    11Ø DIM A (5Ø), B(2Ø,2Ø)
    12Ø DIM Z (5,2Ø)
    13Ø DIM S (5,25)
    14Ø DIM R (4,4)
```

GENERAL FORM:

statement number DIM matrix variable (integer) . . .

or

statement number DIM matrix variable (integer , integer) . . .

Purpose

Reserves working space in memory for a matrix.

The maximum *integer* value (matrix bound) is 255.

Comments

The integers refer to the number of matrix elements if only one dimension is supplied, or to the number of rows and columns respectively, if two dimensions are given.

A matrix (array) variable is any single letter from A to Z.

Arrays not mentioned in a DIM statement are assumed to have 1Ø elements if one-dimensional, or 1Ø rows and columns if two-dimensional.

The working size of a matrix may be smaller than its physical size. For example, an array declared 9 x 9 in a DIM statement may be used to store fewer than 81 elements; the DIM statement supplies only an upper bound on the number of elements.

The absolute maximum matrix size depends on the memory size of the computer.

4.1.3 MAT...ZER

EXAMPLES:

```
305 MAT A = ZER
310 MAT Z = ZER (N)
315 MAT X = ZER (30, 10)
320 MAT R = ZER (N, P)
```

GENERAL FORM:

statement number MAT *matrix variable* = ZER

or

statement number MAT *matrix variable* = ZER (*expression*)

or

statement number MAT *matrix variable* = ZER (*expression* , *expression*)

Purpose

Sets all elements of the specified matrix equal to 0; a new working size may be established.

Comments

The new working size in a MAT . . . ZER is an implicit DIM statement, and may not exceed the limit set by the DIM statement on the total number of elements in an array.

Since Ø has a logical value of "false," MAT . . . ZER is useful in logical initialization.

4.1.4 MAT . . . CON

EXAMPLES:

```
2Ø5 MAT C = CON
21Ø MAT A = CON (N,N)
22Ø MAT Z = CON (5,2Ø)
23Ø MAT Y = CON (5Ø)
```

GENERAL FORM:

statement number MAT *matrix variable* = CON

or

statement number MAT *matrix variable* = CON (*expression*)

or

statement number MAT *matrix variable* = CON
(*expression* , *expression*)

Purpose

Sets up a matrix with all elements equal to 1; a new working size may be specified, within the limits of the original DIM statement on the total number of elements.

Comments

The new working size (an implicit DIM statement) may be omitted as in example statement 2Ø5.

Note that since 1 has a logical value of "true," the MAT . . . CON statement is useful for logical initialization.

The expressions in new size specifications should evaluate to integers. Non-integers are rounded to the nearest integer value.

4.2 INPUTTING SINGLE MATRIX ELEMENTS

EXAMPLES:

```
6ØØ INPUT A(5)
61Ø INPUT B(5,8)
62Ø INPUT R(X), N, A(3,3) , S ,T
63Ø INPUT Z(X,Y), P3, W
64Ø INPUT Z(X,Y), Z(X+1, Y+1), Z(X+R3, Y+S2)
```

GENERAL FORM:

statement number INPUT *matrix variable (expression) . . .*

or

statement number INPUT *matrix variable*
(expression , expression) . . .

Purpose

Allows input of a specified matrix element from the teleprinter.

Comments

The subscripts (in *expressions*) used after the matrix variable designate the row and column of the matrix element. Do not confuse these expressions with working size specifications, such as those following a MAT READ statement.

Expression used as subscripts should evaluate to integers. Non-integers are rounded to the nearest integer value.

Inputting, printing, and reading individual array elements are logically equivalent to simple variables and may be intermixed in INPUT, PRINT, and READ statements.

4.3 PRINTING

4.3.1 PRINTING SINGLE MATRIX ELEMENTS

EXAMPLES:

```
8ØØ PRINT A(3)
81Ø PRINT A (3,3);
82Ø PRINT F(X); E; C5; R(N)
83Ø PRINT G(X,Y)
84Ø PRINT Z(X,Y), Z(1,5), Z(X+N), Z(Y+M)
```

GENERAL FORM:

statement number PRINT *matrix variable (expression) . . .*

or

statement number PRINT *matrix variable*
(expression , expression) . . .

Purpose

Causes the specified matrix element(s) to be printed.

Comments

Expressions used as subscripts should evaluate to integers. Non-integers are rounded to the nearest integer value.

A trailing semicolon packs output into twelve elements per teleprinter line, if possible (statement 81Ø above). A trailing comma or RETURN prints five elements per line.

Expressions (or subscripts) following the matrix variable designate the row and column of the matrix element. Do not confuse these with new working size specifications, such as those following a MAT IDN (identity matrix) statement.

4.3.2 MAT PRINT

EXAMPLES:

```
500 MAT PRINT A
5Ø5 MAT PRINT A;
515 MAT PRINT A,B,C
52Ø MAT PRINT A,B,C;
```

GENERAL FORM:

statement number MAT PRINT *matrix variable*

or

statement number MAT PRINT *matrix variable* , *matrix variable . . .*

Purpose

Causes an entire matrix to be printed, row by row, with double spacing between rows.

Comments

Matrices may be printed in "packed" rows up to 12 elements wide by using the ";" separator, as in example statement 5Ø5. Separation with commas or a RETURN prints 5 elements per row.

4.4 READING

4.4.1 READING MATRIX ELEMENTS

EXAMPLES:

```
9ØØ READ A(6)
91Ø READ A(9,9)
92Ø READ C(X); P; R7
93Ø READ C(X,Y)
94Ø READ Z(X,Y), P(R2, S5), X(4)
```

GENERAL FORM:

statement number READ *matrix variable* (*expression*)

or

statement number READ *matrix variable*
(*expression* , *expression*) . . .

Purpose

Causes the specified matrix element to be read from the current DATA statement.

Comments

Expressions (used as subscripts) should evaluate to integers. Non-integers are rounded to the nearest integer.

Expressions following the matrix variable designate the row and column of the matrix element. Do not confuse these with working size specifications, such as those following MAT READ statement.

The MAT READ statement is used to read an entire matrix from DATA statements. See details in this section.

4.4.2 MAT READ

EXAMPLES:

```
    35Ø MAT READ A
    37Ø MAT READ B(5),C,D
    38Ø MAT READ Z (5,8)
    39Ø MAT READ N (P3,Q7)
```

GENERAL FORM:

statement number MAT READ *matrix variable*

or

statement number MAT READ *matrix variable (expression)*. . .

or

statement number MAT READ *matrix variable*
(expression , expression). . .

Purpose

Reads an entire matrix from DATA statements. A new working size may be specified, within the limits of the original DIM statement.

Comments

MAT READ causes the entire matrix to be filled from the current DATA statement in the row, column order: 1,1; 1,2; 1,3; etc. In this case, the DIM statement controls the number of elements read.

4.5 MATRIX ARITHMETIC

4.5.1 MATRIX ADDITION

EXAMPLES:

```
    31Ø MAT C = B + A
    32Ø MAT X = X + Y
    33Ø MAT P = N + M
```

GENERAL FORM:

statement number MAT *matrix variable = matrix variable + matrix variable*

Purpose

Establishes a matrix equal to the sum of two matrices of identical dimensions; addition is performed element by element.

Comments

The resulting matrix must be previously mentioned in a DIM statement if it has more than 1Ø elements, or 1Ø x 1Ø elements if two-dimensional. Dimensions must be the same as the operand matrices.

The same matrix may appear on both sides of the = sign, as in example statement 32Ø.

4.5.2 MATRIX SUBTRACTION

EXAMPLES:

```
550 MAT C = A - B
560 MAT B = B - Z
570 MAT X = X - A
```

GENERAL FORM:

statement number MAT *matrix variable = matrix variable – matrix variable*

Purpose

Establishes a matrix equal to the difference of two matrices of identical dimensions; subtraction is performed element by element.

Comments

The resulting matrix must be previously mentioned in a DIM statement if it has more than 1Ø elements, or 1Ø x 1Ø elements if two-dimensional. Its dimension must be the same as the operand matrices.

The same matrix may appear on both sides of the = sign, as in example statement 56Ø.

4.5.3 MATRIX MULTIPLICATION

EXAMPLES:

```
930 MAT Z = B * C
940 MAT X = A *A
950 MAT C = Z * B
```

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable* * *matrix variable*

Purpose

Establishes a matrix equal to the product of the two specified matrices.

Comments

Following the rules of matrix multiplication, if the dimensions of matrix B = (P,N) and matrix C = (N,Q), multiplying matrix B by matrix C results in a matrix of dimensions (P,Q).

Note that the product matrix must have an appropriate working size.

The same matrix variable may not appear on both sides of the = sign.

4.5.4 SCALAR MULTIPLICATION

EXAMPLES:

```
110 MAT A = (5) * B
115 MAT C = (10) * C
120 MAT C = (N/3) * X
130 MAT P = (Q7*N5) * R
```

GENERAL FORM:

statement number MAT *matrix variable =*
*(expression) * matrix variable*

Purpose

Establishes a matrix equal to the product of a matrix multiplied by a specified expression (number); that is, each element of the original matrix is multiplied by the number.

Comments

The resulting matrix must be previously mentioned in a DIM statement if it contains more than 10 elements (10 x 10 if two-dimensional).

The same matrix variable may appear on both sides of the = sign.

Both matrices must have the same working size.

4.6 COPYING A MATRIX

EXAMPLES:

```
405 MAT B = A
410 MAT X = Y
420 MAT Z = B
```

GENERAL FORM;

statement number MAT *matrix variable* = *matrix variable*

Purpose

Copies a specified matrix into a matrix of the same dimensions; copying is performed element by element.

Comments

The resulting matrix must be previously mentioned in a DIM statement if it has more than 1Ø elements, or 1Ø x 1Ø if two-dimensional. It must have the same dimensions as the copied matrix.

4.7 IDENTITY MATRIX

EXAMPLES:

```
205 MAT A = IDN
21Ø MAT B = IDN (3,3)
215 MAT Z = IDN (Q5, Q5)
22Ø MAT S = IDN (6, 6)
```

GENERAL FORM:

statement number MAT *array variable* = IDN

or

statement number MAT *array variable* = IDN
(expression , expression)

Purpose

Establishes an identity matrix (all Ø's, with a diagonal from left to right of all 1's); a new working size may be specified.

Comments

The IDN matrix must be two-dimensional and square.

Specifying a new working size has the effect of a DIM statement.

```
Sample identity matrix:  1  Ø  Ø
                         Ø  1  Ø
                         Ø  Ø  1
```

4.8 MATRIX MANIPULATION

4.8.1 MATRIX TRANSPOSITION

EXAMPLES:

```
959 MAT Z = TRN (A)
969 MAT X = TRN (B)
979 MAT Z = TRN (C)
```

GENERAL FORM:

statement number MAT *matrix variable* = TRN
(*matrix variable*)

Purpose

Establishes a matrix as the transposition of a specified matrix (transposes rows and columns).

Comments

Sample transposition:

Original	Transposed
1 2 3	1 4 7
4 5 6	2 5 8
7 8 9	3 6 9

Note that the dimensions of the resulting matrix must be the reverse of the original matrix. For instance, if A has dimensions of 6,5 and MAT C = TRN (A), C must have dimensions of 5,6.

Matrices cannot be transposed or inverted into themselves.

4.8.2 MATRIX INVERSION

EXAMPLES:

```
380 MAT A = INV(B)
390 MAT C = INV(A)
400 MAT Z = INV(Z)
```

GENERAL FORM:

statement number MAT *matrix variable* = INV
(*matrix variable*)

Purpose

Establishes a square matrix as the inverse of the specified square matrix of the same dimensions.

Comments

The inverse is the matrix by which you multiply the original matrix to obtain an identity matrix.

For example:

$$\begin{array}{ccccc} \text{Original} & & \text{Inverse} & & \text{Indentity} \\ \begin{pmatrix} 1 & \text{Ø} & \text{Ø} \\ 1 & 1 & \text{Ø} \\ 1 & 1 & 1 \end{pmatrix} & \text{X} & \begin{pmatrix} 1 & \text{Ø} & \text{Ø} \\ -1 & 1 & \text{Ø} \\ \text{Ø} & -1 & 1 \end{pmatrix} & = & \begin{pmatrix} 1 & \text{Ø} & \text{Ø} \\ \text{Ø} & 1 & \text{Ø} \\ 0 & \text{Ø} & 1 \end{pmatrix} \end{array}$$

Number representation in BASIC is accurate to 6-7 decimal digits; matrix elements are rounded accordingly.

LOGICAL OPERATIONS 5

5.1 LOGICAL VALUES AND NUMERIC VALUES

A distinction should be made between logical values and the numeric values produced by logical evaluation, when using the logical capability of BASIC.

The *logical value* of an expression is determined by definitions established in the user's program.

The *numeric values produced by logical evaluation* are assigned by BASIC. The user may not assign these values.

Logical value is the value of an expression or statement, using the criteria:

any non-zero expression value = "true"

any expression value of zero = "false"

When an expression or statement is logically evaluated, it is assigned one of two numeric values, either:

1, meaning the expression or statement is "true,"

or

Ø, meaning the expression or statement is "false."

5.2 RELATIONAL OPERATORS

There are two ways to use the relational operators in logical evaluations:

1. As a simple check on the numeric value of an expression.

EXAMPLES:

```
150 IF B=7 THEN 6ØØ
2ØØ IF A9#27.65 THEN 7ØØ
3ØØ IF (Z/1Ø)>Ø THEN 8ØØ
```

When a statement is evaluated, if the IF condition is currently "true" (for example, B = 7 in statement 15Ø), then control is transferred to the specified statement; if it is not "true," control passes to the next statement in the program.

Note that the numeric value produced by the logical evaluation is unimportant when the relational operators are used in this way. The user is concerned only with the presence or absence of the condition indicated in the IF statement.

2. As a check on the numeric value produced by logically evaluating an expression, that is: "true" = 1, "false" = Ø.

EXAMPLES:

```
61Ø LET X=27
615 PRINT X=27
62Ø PRINT X#27
63Ø PRINT X>=27
```

The example PRINT statements give the numeric values produced by logical evaluation. For instance, statement 615 is interpreted by BASIC as "Print 1 if X equals 27, Ø if X does not equal 27." There are only two logical alternatives; 1 is used to represent "true," and Ø "false."

The numeric value of the logical evaluation is dependent on, but distinct from, the value of the expression. In the example above, X equals 27, but the numeric value of the logical expression X=27 is 1 since it describes a "true" condition.

5.3 BOOLEAN OPERATORS

There are two ways to use the Boolean Operators.

1. As logical checks on the value of an expression or expressions.

EXAMPLES:

```
51Ø IF A1 OR B THEN 67Ø
52Ø IF B3 AND C9 THEN 68Ø
53Ø IF NOT C9 THEN 69Ø
54Ø IF X THEN 7ØØ
```

Statement 51Ø is interpreted: "If either A1 is 'true' (has a non-zero value) or B is 'true' (has a non-zero value), then transfer control to statement 67Ø."

Similarly, statement 54Ø is interpreted: "If X is 'true' (has a non-zero value), then transfer control to statement 7ØØ.

The Boolean operators evaluate expressions for their logical values only: these are "true" = any non-zero value, "false" = zero. For example, if B3 = 9 and C9 = –5, statement 52Ø would evaluate to "true," since both B3 and C9 have a non-zero value.

2. As a check on the numeric value produced by logically evaluating an expression, that is: "true" = 1, "false" = Ø.

EXAMPLES:

```
49Ø LET B = C = 7
5ØØ PRINT B AND C
51Ø PRINT C OR B
52Ø PRINT NOT B
```

Statements 5ØØ – 52Ø return a numeric value of either 1, indicating that the statement has a logical value of "true," or Ø, indicating a logical value of "false."

Note that the criteria for determining the logical values are:

"true" = any non-zero expression value.

"false" = an expression value of Ø.

The numeric value 1 or Ø is assigned accordingly.

5.4 SOME EXAMPLES

These examples show some of the possibilities for combining logical operators in a statement.

It is advisable to use parentheses wherever possible when combining logical operators.

EXAMPLES:

```
310 IF (A9 AND B7)=Ø OR (A9 + B7)>1ØØ THEN 9ØØ
310 PRINT (A>B) AND (X<Y)
32Ø LET C = NOT D
33Ø IF (C7 OR D4) AND (X2 OR Y3) THEN 93Ø
34Ø IF (A1 AND B2) AND (X2 AND Y3) THEN 94Ø
```

The numerical value of "true" or "false" may be used in algebraic operations. For example, this sequence counts the number of zero values in DATA statements.

```
 .
 .
 9Ø LET X = Ø
1ØØ FOR I = 1 TO N
11Ø READ A
12Ø LET X = X+(A=Ø)
13Ø NEXT I
14Ø PRINT N; "VALUES WERE READ."
15Ø PRINT X; "WERE ZEROS."
16Ø PRINT (N-X); "WERE NON-ZERO."
  .
  .
  .
```

Note that X is increased by 1 or Ø each time A is read; when A= Ø, the expression A = Ø is "true," and X is increased by 1.

SYNTAX REQUIREMENTS OF BASIC 6

LEGEND

::=	"is defined as . . ."
|	"or"
< >	enclose an element of BASIC

LANGUAGE RULES

1. The <COM statement>, if any exists, must be the first statement presented and have the lowest sequence number; the last statement must be an <END statement>.

2. A sequence number may not exceed 9999 and must be non-zero.

3. Exponent integers may not have more than two digits.

4. A formal bound may not exceed 255 and must be non-zero.

5. A subroutine number must lie between 1 and 63, inclusive.

6. Strings may not contain the quote character (").

7. A <bound part> for an IDN must be doubly subscripted.

8. An array may not be inverted or transposed into itself.

9. An array may not be replaced by itself multiplied by another array.

```
< basic program >           ::= < program statement > | < basic pro-
                                gram > < program statement >(1)
< program statement >       ::= < sequence number > < basic state-
                                ment > carriage return
< sequence number >         ::= < integer >(2)
< basic statement >         ::= < let statement > | < dim
                                statement > | < com statement > |
                                < def statement > | < rem
                                statement > | < go to statement > |
                                < if statement > | < for statement > |
                                < next statement > |
                                < gosub statement > | < return
                                statement > | < end statement > |
                                < stop statement > | < wait
                                statement > | < call statement > |
                                < data statement > | < read
                                statement | < restore statement > |
                                < input statement > | < print
                                statement > | < mat statement >
< let statement >           ::= <let head > < formula >
< let head >                ::= LET < variable >=| < let head >
                                < variable > =
< formula >                 ::= < conjunction > | < formula > OR
                                < conjunction >
< conjunction >             ::= < Boolean primary > | < conjunction >
                                AND < Boolean primary >
< Boolean primary >         ::= < arithmetic expression > |
                                < Boolean primary >
                                < relational operator > < arithmetic
                                expression >
< arithmetic expression >   ::= < term > | <arithmetic expression >
                                + < term > | < arithmetic expression>
                                - < term >
< term >                    ::= < factor > | < term > * < factor > |
                                < term > / < factor >
< factor >                  ::= < primary > | <sign > < primary > |
                                NOT < primary >
```

```
< primary >                  ::= < operand > | < primary > ↑
                                 < operand >
< relational operator >      ::= > | < | > = | < = | = | # | < >
< operand >                  ::= < variable > | < unsigned number > |
                                 < system function > | < function > |
                                 < formula operand >
< variable >                 ::= < simple variable > | < subscripted
                                 variable >
< simple variable >          ::= < letter > | < letter > < digit >
< subscripted variable >     ::= < array identifier > < subscript head>
                                 < subscript > < right bracket >
< array identifier >         ::= < letter >
< subscript head >           ::= < left bracket > | < left bracket >
                                 < subscript >
< subscript >                ::= < formula >
< letter >                   ::= 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
< digit >                    ::= Ø | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
< left bracket >             ::= ( | [
< right bracket >            ::= ) | ]
< sign >                     ::= + | -
< unsigned number >          ::= < decimal part > | < decimal part >
                                 < exponent >
< decimal part >             :: = < integer > | < integer > . < integer >
                                  | . < integer >
< integer >                  ::= < digit > | < integer > < digit >
< exponent >                 ::= E < integer > | E < sign >
                                 < integer >(3)
< system function >          ::= < system function name >
                                 < parameter part >
< system function name >     ::= SIN | COS | TAN | ATN | EXP |
                                 LOG | ABS | SQR | INT | RND | SGN
< parameter part >           ::= < left bracket > < actual parameter >
                                 < right bracket >
< actual parameter >         ::= < formula >
```

< function > ::= FN < letter > < parameter part >

< formula operand > ::= < left bracket > < formula > < right bracket >

< dim statement > ::= DIM < formal array list >

< formal array list > ::= < formal array > | < formal array list > , < formal array >

< formal array > ::= < array identifier > < formal bound head > < formal bound > < right bracket >

< formal bound head > ::= < left bracket > | < left bracket > < formal bound >,

< formal bound > ::= < integer >(4)

< com statement > ::= COM < formal array list >

< def statement > ::= DEF FN < letter > < left bracket > < formal parameter > < right bracket > = < formula >

< formal parameter > ::= < simple variable >

< rem statement > ::= REM < character string >

< character string > ::= any teletype character except carriage return, alt mode, escape, rubout, line feed, null, control B, control C, or left arrow

< goto statement > ::= GO TO < sequence number >

< if statement > ::= IF < formula > THEN < sequence number >

< for statement > ::= < for head > | < for head > STEP < step size >

< for head > ::= FOR < for variable > = < initial value > TO < limit value >

< for variable > ::= < simple variable >

< initial value > ::= < formula >

< limit value > ::= < formula >

< step size > ::= < formula >

< next statement > ::= NEXT < for variable >

< gosub statement > ::= GOSUB < sequence number >

< return statement > ::= RETURN

< end statement > ::= END

< stop statement > ::= STOP

< wait statement > ::= WAIT < parameter part >

< call statement > ::= CALL < call head > < right bracket >

< call head > ::= < left bracket > < subroutine number > | < call head > , < actual parameter >

< subroutine number > ::= < integer >(5)

< data statement > ::= DATA < constant > | < data statement > , < constant >

< constant > ::= < unsigned number > | < sign > < unsigned number >

< read statement > ::= READ < variable list >

< variable list > ::= < variable > | < variable list > , < variable >

< restore statement > ::= RESTORE

< input statement > ::= INPUT < variable list >

< print statement > ::= < print head > | < print head > < print formula >

< print head > ::= PRINT | < print head > < print part >

< print part > ::= < string > | < string > < delimiter > | < print formula > < delimiter > | < print formula > < string > | < print formula > < string > < delimiter >

< string > ::= “ < character string > ”(6)

< delimiter > ::= , | ;

```
< print formula >             ::= < formula > | TAB < parameter
                                  part >
< mat statement >             ::= MAT < mat body >
< mat body >                  ::= < mat read > | < mat print > |
                                  < mat replacement >
< mat read >                  ::= READ < actual array > | < mat
                                  read > , < actual array >
< actual array >              ::= < array identifier > | < array
                                  identifier > < bound part >
< bound part >                ::= < actual bound head >
                                  < actual bound > < right bracket >
< actual bound head >         ::= < left bracket > | < left bracket >
                                  < actual bound > ,
< actual bound >              ::= < formula >
< mat print >                 ::= PRINT < mat print part > | PRINT
                                  < mat print part > < delimiter >
< mat print part >            ::= < array identifier > | < mat print
                                  part > < delimiter > < array
                                  identifier >
< mat replacement >           ::= < array identifier > = < mat
                                  formula >
< mat formula >               ::= < array identifier > | < mat
                                  function > | < array identifier >
                                  < mat operator > < array
                                  identifier > | < formula operand >*
                                  < array identifier >
< mat function >              ::= < mat initialization > | < mat
                                  initialization > < bound part > |
                                  INV < array parameter > |
                                  TRN < array parameter >
< mat initialization >        ::= ZER | CON | IDN (7)
< array parameter >           ::= < left bracket > < array identifier >
                                  < right bracket > (8)
< mat operator >              ::= + | – | * (9)
```

FOR ADVANCED PROGRAMMERS 7

7.1 MODIFYING HP BASIC

As indicated in the configuration instructions, an HP BASIC system configured with PBS may include user-written assembly language subroutines. These subroutines are accessed with a CALL statement while a BASIC program is running. HP BASIC may also be run under the HP Magnetic Tape System (MTS), provided that the amount of core memory in the configured tape of HP BASIC is the same as the MTS under which it is run.

The information in this section is intended to help the programmer in modifying HP BASIC. Users are reminded that HP cannot be responsible for non-standard or user-modified software.

7.2 CALL STATEMENT

EXAMPLE:

```
20 CALL (5, A(10), 1, 1188, 10)
```

GENERAL FORM:

statement number CALL *(statement number , parameter list)*

Purpose

Allows addition of absolute assembly language routines (such as input-output drivers) to BASIC, for specialized configurations. CALL transfers control to the specified assembly language subroutine.

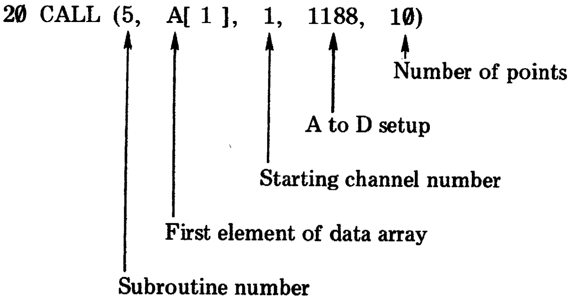
Comments

Subroutines executed by CALL are not constrained by BASIC and have absolute control of the computer. The assembly language subroutine may, therefore, alter any portion of the system, including BASIC. For this reason, it is recommended that only programmers proficient in assembly language attempt to add CALL subroutines to BASIC programs.

CALL subroutines are "loaded into the computer" through the photoreader or terminal tape reader either at configuration time or as a load-time overlay.

The CALL *subroutine number* is a positive integer between 1 and 63 specifying the desired subroutine. If no such subroutine number exists, the statement is rejected.

The other parameters, separated by commas, may be any formula and their number is dependent upon the subroutine called. For example, a subroutine designated by 5 is appended to the system to take readings from an A to D subsystem and store them in an array. The parameters specify the array into which the values are inserted, the channel number of the first point to be measured, the setup for the A to D converter and the number of points to be measured. A representative call for this subsystem is:



```
20 CALL (5,  A[ 1 ],  1,  1188,  10)
```

When using the CALL statement, it is important that correct parameters be specified. Accidentally reversing the first and second parameters could destroy the core-resident BASIC system, unless precautions have been taken by the writer of the called subroutine to protect the BASIC system.

The parameters of a CALL statement provide the dynamic link between BASIC and the called subroutine. Prior to transferring control to the subroutine, BASIC evaluates the parameters and stacks the addresses of the results. Upon entering the subroutine, the A-register contains the address of this stack (i.e., the address of the addresses of the parameter values). The A-register initially points to the address of the first parameter; successively decrementing the A-register causes it to point to successive parameter addresses. For example, if the A-register = 17302 when a subroutine is entered, the first parameter address is 17302, the second 17301, the third 17300, etc.

The parameter addresses passed by BASIC give the subroutine access to values in the BASIC program. The only way a called subroutine can transmit results to a BASIC program is to store them by means of a parameter address.

Transmittal of quantities of data between a BASIC program and a called subroutine is most conveniently handled through arrays. Since only addresses are passed to a subroutine, an array parameter must be an element of the array (in general this would be the first element of the array). It is important to remember that arrays are stored by rows, and that each element is a floating point number occupying two (16-bit) words. Hence, if an array A has M columns per row, the address of A[I,J] is (address A[1,1]+ 2(M(I–1) + (J–1)).

To output from a subroutine to the terminal:

1. Load a buffer address into the B-register.
2. Load a character count into the A-register.
3. Execute a JSB 102B, I.

The referenced block of core is then interpreted as an ASCII string and output, followed by a RETURN *linefeed* if the count was negative.

Whenever data is transferred from a called subroutine through the address of a parameter, there is a danger that the BASIC system or a program might be destroyed. This situation can arise when parameters are specified incorrectly or insufficient space is allocated in a data array. For example, constants such as 2 or –1.1 in a BASIC program are stored in the program as they appear; therefore, storing through the address of a constant parameter changes the actual constant in the CALL statement. A subsequent execution of that statement may lead to unpredictable results. A parameter that is an expression (for example, A AND B or NOT A AND B) is evaluated and the result placed in a temporary location. Since the parameter address references this temporary location, storing into it will not harm the BASIC system or program. However, the value stored there is lost to the BASIC program. If a called subroutine stores more values in an array than the array can hold, the resulting overflow of data may destroy the BASIC system or program.

Users of CALL statements should be cautioned against using unsuitable parameters in CALL statements (especially against using a simple variable or a constant where an array element is expected). Also, when using arrays as parameters it is good practice to include the dimensions of the array as additional parameters to allow a means of checking within the subroutine.

An effective protection requires additional programming effort. BASIC contains sets of pointers delimiting the areas of memory within which different types of parameters exist. By checking parameter addresses against these bounds, the subroutine can verify that they are of the expected type. If X represents the parameter address, the following applies:

a. Constant parameter $(112_8) < X < (113_8)$

b. Simple variable parameter $(116_8) < X < (117_8)$

c. Array parameter

1) In common storage $(110_8) \leqslant X < (112_8)$

2) Not in common storage $(113_8) \leqslant X < (115_8)$

d. Expression parameter $(115_8) < X < (120_8)$
where (112_8) means the contents of location number octal 112.

7.3 BYE COMMAND

EXAMPLE: BYE

GENERAL FORM: BYE

Purpose

Produces a HLT 77_8 when used under the HP BASIC system; or causes transfer of control from the HP BASIC system to the Magnetic Tape System (MTS) executive when used in an MTS based HP BASIC system.

Comments

HP BASIC may be configured as part of an HP Magnetic Tape System.

If it is intended to run under the Magnetic Tape System, PBS may be configured separately or together with the HP BASIC interpreter.

User-written assembly language subroutines may be added to an MTS based HP BASIC system; they may be configured along with the drivers and interpreter using PBS or added while preparing the MTS.

Note that configuration of an HP BASIC system cannot be done under the control of an MTS; rather a configured system may be one of the subsystems supplied when creating an MTS.

Remember that an HP BASIC system running under MTS must specify the same core memory size as the MTS.

7.4 FIRST AND LAST WORDS OF AVAILABLE MEMORY

The first word of available memory (FWAM) is contained in location 110_8 in the HP BASIC system.

The last word of available memory (LWAM) is contained in location 111_8 in the HP BASIC system.

Comments

When HP BASIC is run under MTS, FWAM is contained in location 11Ø$_8$; LWAM is dynamically determined and placed in location 1Ø6$_8$ after the system is loaded.

7.5 FIRST WORD AVAILABLE IN BASE PAGE

The address of the first word available in base page is contained in location 114_8. All locations from the location referenced in 114_8 through 1777_8 are not used by BASIC, and are therefore available for CALL subroutines or other modifications.

7.6 LINK POINTS

For ease in user modification, locations 2Ø1$_8$ through 322$_8$ contain links to various subportions and subroutines of BASIC often used in creating customized systems. The identity and locations of these links is fixed (will not change with subsequent versions), but the contents of these locations are subject to change if the routines they point to move as a result of future revisions. The assembly language listing of the HP BASIC interpreter captions each link briefly. Since these links are an integral part of BASIC, the user is responsible for interpreting and using this information.

7.7 LINKAGES TO SUBROUTINES

BASIC accesses called subroutines through a table containing linkage information. Entries in the table, one per subroutine, are two words in length. Bits 5-Ø of the first word contain the number identifying the subroutine (chosen freely from 1 to 77$_8$ inclusive) and bits 15-8 contain the number of parameters passed to the subroutine. (CALL statements with an incorrect number of parameters are rejected by the syntax analyzer.) The second word contains the absolute address of the entry point of the subroutine. (Control is transferred via a JSB.) Although subroutine numbers need not be assigned in any particular order, all entries in the table must be contiguous. An acceptable auxiliary tape contains the following:

1. An ORG statement to originate the program at an address greater than that of the last word of the BASIC system. The address of this last word + 1 is contained in location 11Ø$_8$ of the standard BASIC system. Hence, a suitable lower limit for the origin address can be determined by loading BASIC and examining location 11Ø$_8$.

2. The subroutine linkage table described above.

3. The assembly language subroutines.

4. Code to set the following linkage addresses:

   a. In location 11Ø$_8$ put the address of the last word + 1 used in the auxiliary tape.

   b. In location 121$_8$ put the address of the first word of the subroutine linkage table.

   c. In location 122$_8$ put the address of the last word + 1 of the subroutine linkage table.

Assuming, for example, that location 11Ø$_8$ of the standard BASIC system contains 13142$_8$; an acceptable auxiliary tape could be assembled from the following code:

```
          ORG  13142B
SBTBL     OCT  24Ø6        Subroutine 6 has 5 parameters
          DEF  SB6
          OCT  1421        Subroutine 17 has 3 parameters
          DEF  SB17
ENDTB     EQU  *
SB6       NOP
           .
           .               Subroutine #6 body
           .
          JMP  SB6,I
SB17      NOP
           .
           .               Subroutine #17 body
           .
          JMP  SB17, I
LSTWD     EQU  *
          ORG  11ØB
```

```
DEF   LSTWD
ORG   121B
DEF   SBTBL
DEF   ENDTB
END
```

Acceptable calls to subroutines SB6 and SB17 might be

```
CALL (6, A, B, 1, N*3, SIN(X+Y))
CALL (17, A[1], 5, N)
```

NOTE: Location 111_8 of the standard BASIC system contains the address of the last word of available memory. It is not possible to create a system which requires more space than that existing between the addresses in locations $11Ø_8$ and 111_8. Systems using all or most of this space leave very little space for the user of the system.

7.8 DELETING THE MATRIX SUBROUTINES

This assembly language pseudo-program shows a method of deleting the MAT execution package to gain more user space, or for replacing it with CALL routines or other customized code.

```
ORG <contents of 21Ø8>
OCT Ø,Ø
ORG 11ØB
DEF <contents of 2118>
```

This sequence has the effect of preventing the syntax processor from recognizing "MAT" and of resetting the first word of available memory pointer to the first word of the matrix execution package.

GENERATING HP BASIC A

An HP BASIC system consists of the HP BASIC interpreter and the Prepare BASIC System (PBS) programs. Assembly language subroutines written by the user may be included.

The HP BASIC tape consists of the HP BASIC interpreter. The PBS tape contains drivers for the terminal, photoreader, high-speed punch, and the routines necessary to configure these drivers into an HP BASIC system.

An HP BASIC system is generated by:

- Loading the configuration program (PBS) into memory.
- Loading other tapes (HP BASIC, user subroutines) to be included on the system tape.
- Using PBS to configure the HP BASIC System and to dump it onto a single tape.
- Loading the configured HP BASIC System tape into memory along with any separate programs (HP BASIC, user subroutines) included in the system.

CONFIGURING AN HP BASIC SYSTEM

1. Decide which elements the configured HP BASIC system tape will contain.

   The three choices are:

   a. I/O drivers, BASIC, user subroutines

   b. I/O drivers, BASIC

   c. I/O drivers.

2. Turn on all necessary peripheral devices (teleprinter, tape punch, etc).

3. Make sure the computer has halted.

4. Use the Basic Binary Loader (BBL) or the Basic Binary Disc Loader (BBDL) to load the PBS tape into memory.*

5. If option a or b was chosen in step 1, use the BBL or the BBDL to load the HP BASIC tape into memory. If option a or b was not chosen, skip to step 7.

6. If option a was chosen in step 1, then use the BBL or BBDL to load the user-subroutine tapes into memory. If option a was not chosen, skip to step 7.

7. Set a starting address of 2_8.

8. Set the switch register to the octal select code of the terminal. (Set bit 15 OFF.)

9. Start PBS execution.

10. The PBS program types:

    PHOTOREADER I/O ADDRESS?

    Type the photoreader octal select code on the teleprinter keyboard, then press the RETURN key. If there is no photoreader, then press RETURN key only.

**If an operator error is made or if any tape does not load properly, return to step 3 to reload PBS.*

11. PBS then types:

PUNCH I/O ADDRESS?

Type the high-speed punch octal select code on the teleprinter keyboard, then press the RETURN key. If there is no high-speed punch, press the RETURN key only.

12. PBS then asks:

SYSTEM DUMP I/O ADDRESS?

Type the high-speed punch octal select code on the teleprinter keyboard, then press the RETURN key. If no high-speed punch exists, press the RETURN key only.

13. PBS then asks:

CORE SIZE?

Enter the computer core size (8, 16, 24 or 32), then press the RETURN key. (Pressing RETURN only indicates an 8K memory size.)

14. If a high-speed punch is available, a configured HP BASIC system tape is punched. If a high-speed punch is not available, the message:

TURN ON TTY PUNCH, PRESS RUN

is printed, and the computer halts.

15. Turn the teleprinter punch on and start the computer, without modifying the contents of any computer register.

The configured HP BASIC system tape is punched on the teleprinter punch and the computer halts.

16. To punch another copy of the system tape, merely restart the computer without modifying any register contents.

NOTE: After the configured system tape is punched (Steps 14, 15 and 16), the configured system remains intact in memory. To run the system right away on the same computer that configured it, start at Step 4 when using PROCEDURE 2 (to avoid loading in the configured system tape). If the system is to run on a computer different from the one that configured it, or on the same computer at a later time, start at Step 1 when using PROCEDURE 2.

LOADING THE CONFIGURED HP BASIC SYSTEM

1. Turn on all necessary peripheral equipment (teleprinter, tape input device, etc.).

2. Make sure that the computer has halted.

3. Load the configured HP BASIC system tape using the BBL or BBDL.

4. If the HP BASIC interpreter was not included as part of the configured HP BASIC system tape, load the HP BASIC interpreter tape into memory using the BBL or BBDL.

5. If any user subroutines are to be included in the system and if they are not part of the HP BASIC system tape previously loaded, load the user-subroutine tapes using the BBL or BBDL.

6. Set a starting address of 1ØØ$_8$.

7. Start program execution. The message:

   READY

   is typed. HP BASIC is ready for use.

HEWLETT-PACKARD COMPANY, 11000 WOLFE ROAD, CUPERTINO, CALIFORNIA 95014 PRINTED IN U.S.A. 9/72 $3 5951-4423