Doc. No. 130071585C
M-991

HONEYWELL
316/516
PROGRAMMERS REFERENCE MANUAL

May 1969

Honeywell
COMPUTER CONTROL
DIVISION

Original printing Jan. 1968
Revised Aug. 1968 and May 1969

Printed in U.S.A.

Published by the Publications Department,
Honeywell Inc., Computer Control Division

CONTENTS

CONTENTS (Cont)

Page

SECTION II
INSTRUCTION REPERTOIRE

CONTENTS (Cont)

Page

SECTION III
INPUT/OUTPUT CHANNELS AND DEVICES

SECTION IV
DAP-16 LANGUAGE

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

TABLES (Cont)

INTRODUCTION

The Honeywell 316/516 computers are designed for both open-shop scientific applications and real-time on-line data processing and control. Modular design, a flexible I/O structure, and command repertoire enable these machines to be tailored to a broad variety of applications both on- and off-line. These include data reduction, process control, instrumentation, simulations, and open-shop scientific and engineering computation.

General characteristics include fully parallel organization, indexing, multi-level indirect addressing, powerful I/O system, a comprehensive 72-command instruction repertoire, and straightforward logic for easy system interface and field expansion. Selected optional capabilities are designed with plug-in modularity to permit custom tailoring at minimum expense.

The DAP-16 assembler is effective and efficient because it allows the programmer to specify a one- or two-pass assembly for the same source program: one-pass for the basic system and two-pass for systems with high-speed input devices where more detailed listings are required. The programmer can directly address all of memory with his source program through the use of desectorizing software.

DAP-16 provides numerous pseudo-operations to supplement the standard instructions. These pseudo-operations also allow the programmer to express concepts which do not have any counterparts in machine language. Among the important capabilities of these instructions are programmer defined assembly and loader controls, data definitions, and program linkages.

SYSTEM DESCRIPTION

Figure 1, a block diagram of the Honeywell 316/516, shows the data storage registers, the control unit of the central processor, and the input/output controls. The random access memory, shown as a single block, is a magnetic core unit containing one or more memory modules of 4096 or 8192 (DDP-516 only) 16-bit words. Data from the memory is transferred to and from the Honeywell 316/516 registers through the M-register. The functional units of the central processor and the input/output controls are:

A-Register (A): A 16-bit register used as the primary arithmetic and logic register of the computer.

B-Register (B): A 16-bit secondary arithmetic register used primarily to hold arithmetic operands which exceed one word in length.

Program Counter (P): A 16-bit register that contains the location of the next instruction to be executed.

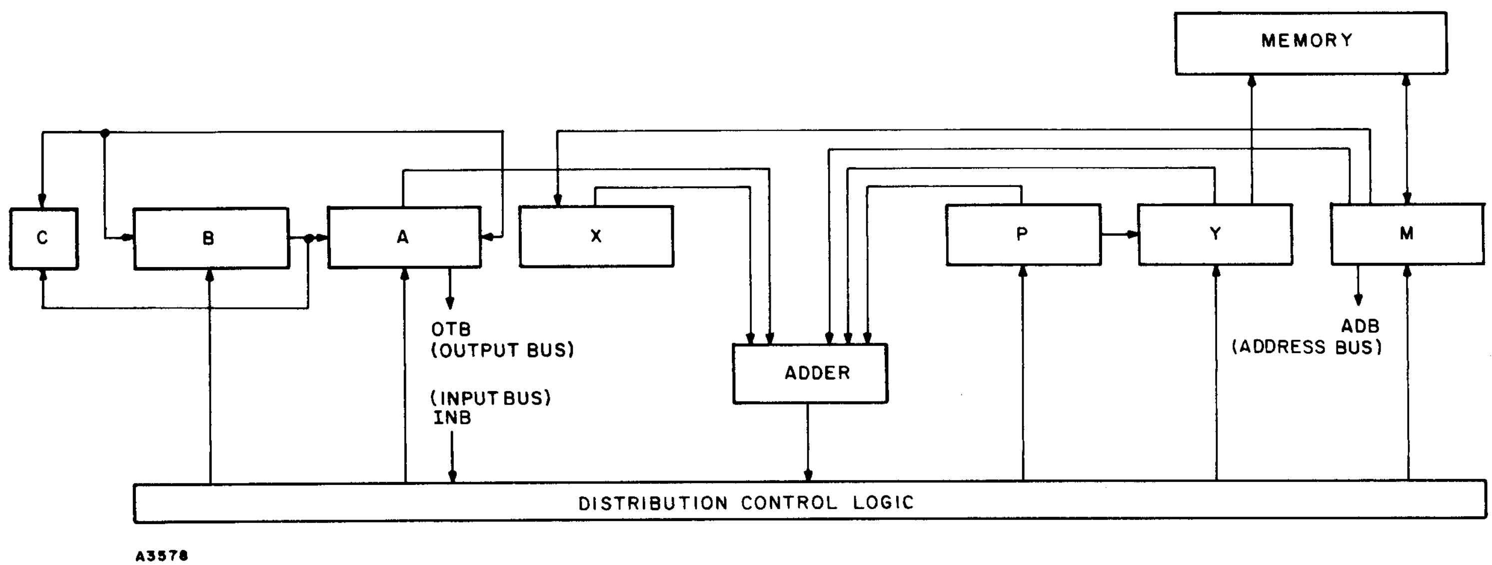


Figure 1. Honeywell 316/516 Simplified Block Diagram

Adder: Performs the basic arithmetic processes of addition and subtraction.

M-Register (M): A 16-bit register used to transfer information to and from the magnetic core memory.

Y-Register (Y): A 16-bit register used to store the address for the memory.

C-Bit (C): A 1-bit indicator associated with the A- and B-registers that stores overflow status resulting from the execution of arithmetic instructions and stores the last bit shifted out of the A- or B-register during the execution of shift instructions.

Index Register (X): A 16-bit register used for address modification. Any memory write cycle addressing memory location zero also loads the X-register.

Output Bus (OTB): Sixteen lines that transmit data from the computer A-register to an I/O device.

Input Bus (INB): Sixteen lines that transmit data from an I/O device to the computer A-register.

Address Bus (ADB): Ten lines used in conjunction with I/O devices. Bits on lines 7 through 10 define the function to be performed by the I/O device. Bits on lines 11 through 16 designate the I/O device to be used.

SPECIFICATIONS

Type

Parallel binary

Addressing

Single address with indexing and indirect addressing

Word Length

16 bits

Machine Code

Two's complement

Memory Type

Magnetic core

Memory Size

4096, 8182, 12,288, 16,384, 24,576 or 32,768 (DDP-516)
4096, 8192, 12,288, 16,384 (H316)

Memory Cycle Time

0.96 μs (DDP-516)
1.6 μs (H316)

Speed

Add: 1.92 μs (DDP-516) 3.2 μs (H316)

Subtract: 1.92 μs (DDP-516) 3.2 μs (H316)

Multiply
(hardware option): 5.28 μs (max)(DDP-516) 8.8 μs (max) (H316)

Divide
(hardware option): 10.56 μs (max)(DDP-516)17.6 μs (max) (H316)

Standard Peripheral Equipment

ASR-33 or -35 Teletype Unit provides the following capabilities:

a. Reads paper tape at 10 characters per second
b. Punches paper tape at 10 characters per second
c. Type at 10 characters per second
d. Keyboard input
e. Off-line paper-tape preparation, reproduction, and listing

Optional Peripheral Equipment

300 characters per second photoelectric paper-tape reader

110 characters per second paper-tape punch

300 line-per-minute (120 character-per-line) high-speed printer

200 card-per-minute card reader

Moving Head Disc File, Model 316/516-4600

Fixed Head Disc File, Model DDP-516-4400

Magnetic tape units:

Unit	Tape Speed (ips)	Density (bpi)
Low Speed	36	200, 556, 800
High Speed	80	200, 556, 800

Standard Input/Output Lines

16-bit input bus

16-bit output bus

10-bit device address bus

External control and sense lines

Input/Output Modes

Three modes are available for data transfer between peripheral devices and the Honeywell 316/516:

a. Single word transfer with or without interrupt
b. Direct multiplex control (DMC) (optional)
c. Direct memory access (DMA) optional (DDP-516 only)

Interrupt

Single interrupt line standard; up to 48 optional priority interrupts are available.

Power Failure Protection

Power failure interrupt standard. Core memory is protected against loss of information on ac power failure.

SECTIONAL DESCRIPTION

The Honeywell 316/516 Programmers Reference Manual is divided into 7 sections and 9 appendices. Programmers should be familiar with all sections including the Introduction which describes information of general interest.

Section I introduces the computer organization while Section II describes the instruction repertoire. The majority of the Honeywell 316/516 I/O channels and devices comprise Section III. The DAP-16 language is discussed in Section IV and the DAP-16 pseudo-operations are detailed in Section V. Section VI contains information on the standard I/O Library and Section VII includes the Honeywell 316/516 mathematical libraries. Appendix A comprises the numbering system and two's complement arithmetic, while Appendix B lists the peripheral device codes. The remaining appendices provide information on standard instructions, main frame option commands, peripheral device commands, dedicated locations, key-in loader, memory map, summary of DAP-16 pseudo-operations, modification of I/O device assignments, and the software package.

PREREQUISITE READING

To complement the understanding of the Honeywell 316/516 Programmers Reference Manual the reader should be familiar with the Honeywell 316/516 Users Guide (Doc. No. 130071627) and FORTRAN IV (Doc. No. 130071364).

SPECIAL INSTRUCTIONS

The Honeywell 316/516 Programmers Reference Manual supersedes the December 1966 edition of the DAP-16 Manual (Doc. No. 130071629) and the DDP-516 Programmers Reference Manual (Doc. No. 130071585).

SECTION I
COMPUTER ORGANIZATION

This section comprises the organizational and functional capabilitites of the Honeywell 316/516 general purpose I/C digital computers. Both hardware and software word formats, memory addressing, indexing, and addressing are described. Instruction sequences, breaks, interrupts, and memory access priority structure are also discussed.

WORD FORMATS (HARDWARE)

Word Structure

Data Words. -- Data words are stored in binary form using two's complement notation. The Honeywell 316/516 accepts and processes data words in both single and double precision. Single precision data words (Figure 1-1) include 15 magnitude bits plus a sign bit and represents a data range of $\pm 2^{15}$ or ±32,768.

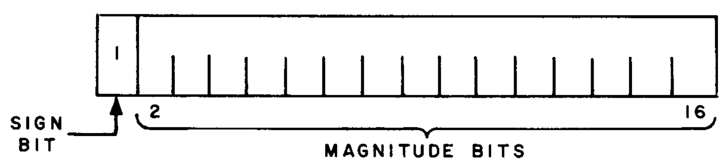


Figure 1-1. Data Word Format, Single Precision

Double precision data words (Figure 1-2) include two data words, each one having 15 magnitude bits. The first data word includes the 15 most significant bits (MSB) of the number plus a sign bit. It is identical to a data word using single precision. The second data word includes the 15 least significant bits (LSB) of the double precision word. The sign position is always zero. Double precision data words represent a data range of $\pm 2^{30}$ or ±1,073,741,824. The High Speed Arithmetic Option (Honeywell 316/516-11) is required for hardware double precision operations.

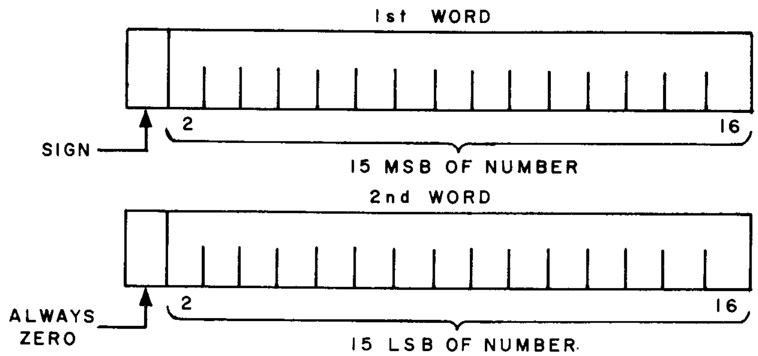


Figure 1-2. Data Word Format, Double Precision

Instruction Words

Instruction words are divided into four types: memory reference, input/output, shift, and generic. The basic instruction word format in the computer is that for a memory reference instruction as shown in Figure 1-3. Bit 1, the flag bit, denotes indirect addressing; bit 2, the tag bit, denotes indexing.* Bits 3 through 6 contain the operation code that defines the function to be performed. For example, if bits 3 through 6 contain 0110 $(06)_8$, the instruction is identified as an ADD instruction; if they contain 1001 $(11)_8$, the instruction is a COMPARE. For ease of communication, operation codes are generally expressed either in octal or as a mnemonic. "Subtract," for example, which has an op code bit configuration of 0111, is referenced in machine language as $(07)_8$ and has a mnemonic of SUB. The latter is the way the programmer writes an op code when programming in DAP-16, the computer's assembly language.

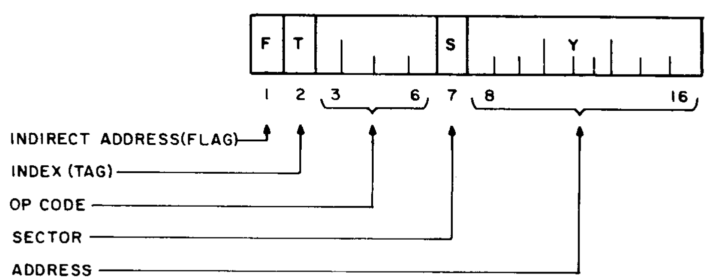


Figure 1-3. Memory Reference Instruction Format

*Bit 2 of an indirect address word in a DDP-516 having more than 16K of core memory is an address bit in the EXTEND mode.

Generic instructions are identified by a word format as shown in Figure 1-4. Bits 1 through 16 denote the op code.

Figure 1-4. Generic Instruction Format

The I/O instruction word format is shown in Figure 1-5. Bits 1 through 6 specify the particular I/O instruction and bits 11 through 16 specify which device is being addressed. Bits 7 through 10 define the function to be performed by the instruction.

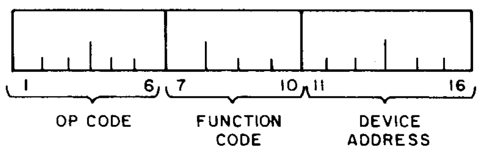


Figure 1-5. Input/Output Instruction Format

The shift instruction word format is shown in Figure 1-6. Bits 1 through 10 specify the type of shift and bits 11 through 16 are used to define the number of shifts to be performed. The number of shifts must be represented in two's complement form.

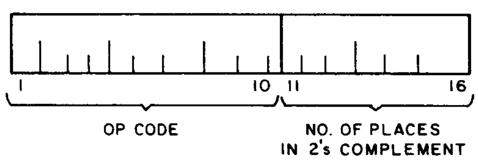


Figure 1-6. Shift Instruction Format

Single Precision

The format for data words stored in the computer is shown in Figure 1-7.

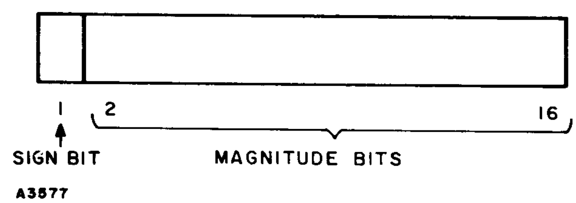


Figure 1-7. Data Word Format, Single Precision

Sixteen-bit data words are stored in two's complement form. The first bit of a data word may be considered the arithmetic sign and is zero for positive data.

Double Precision

When greater precision is required than that obtainable when using the single precision format, the double precision format is used (Figure 1-8). The sign position of the second (least significant) word is always zero. Thirty bits of magnitude are obtainable. This is the format for the product of the multiplication of two single precision words. It is also the data format for double precision operations.

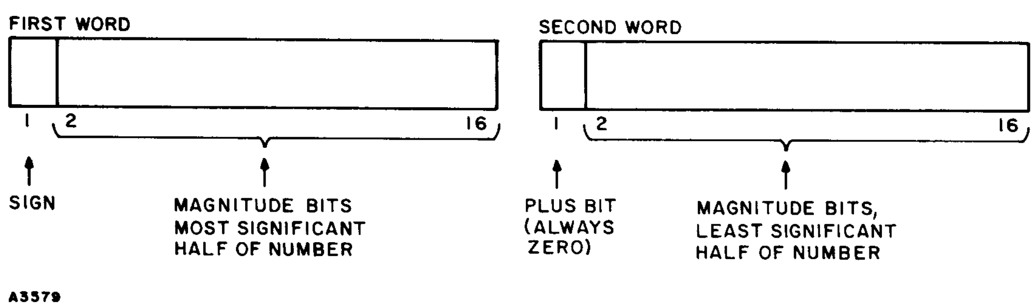


Figure 1-8. Data Word Format, Double Precision

Logical Data

Logical data, such as the condition of 16 binary indicators, can be stored in a single data word. In this case, bit 1 of a word does not represent the sign. This type of data is generally not treated arithmetically by the program but logically by means of Boolean operators such as AND and EXCLUSIVE OR.

WORD FORMATS (SOFTWARE)

Word Format Identifiers

Word formats defined in the next few paragraphs have the following field definitions:

S	=	Sign of number (0 if positive, 1 if negative)
I	=	15-bit integer (2's complement if negative)
E	=	Characteristic of floating-point number (excess 128)
F (1)	=	Most significant bits of fractional part of a normalized floating-point number
F (2) and F (3)	=	16-bit continuation of floating-point fraction
Note:		If the sign bit is negative, the floating-point number is in full 2-word (3-word for double precision) two's complement form.

Integer Format

Following is the format for a right-justified single 15-bit (plus sign) integer. Examples of such integers are:

24 = 000030

-24 = 177750

1															16
S	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I

Figure 1-9. Integer Format

Real Format

Following is the format for a 2-word normalized floating-point number of sign and 23- bit accuracy, and an 8-bit characteristic. Examples of real numbers are:

0.1	=	037346, 063146
503.25	=	042375, 150000
-0.1	=	140431, 014632
-503.25	=	135402, 030000

	1	2							9	10						16
1st WORD	S	E	E	E	E	E	E	E	E	F	F	F	F	F	F	F

	1															16
2nd WORD	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Figure 1-10. Real Format

Double Precision Format

Following is the format for a 3-word normalized floating-point number of sign and 39-bit accuracy, and an 8-bit characteristic. Examples of double precision numbers are:

0.1 = 037346, 063146, 063146

-0.1 = 140431, 114631, 114632

	1	2							9	10						16
1 st WORD (AC1)	S	E	E	E	E	E	E	E	E	F	F	F	F	F	F	F
	1															16
2 nd WORD (AC2)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
	1															16
3 rd WORD (AC3)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Figure 1-11. Double Precision Format

Complex Format

Following is the format for a complex number consisting of two real format arguments, each being a two-word normalized floating-point number of sign and 23-bit accuracy. The first real argument represents the real part of the complex argument, while the second real argument represents the imaginary part of the complex argument. An example of a complex number is:

503.25, -0.1 = 042375, 150000, 140431, 014632

		1	2							9	10						16
ARG. 1	1 st WORD	S	E	E	E	E	E	E	E	E	F	F	F	F	F	F	F
		1															16
	2 nd WORD	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
		1	2							9	10						16
ARG. 2	1 st WORD	S	E	E	E	E	E	E	E	E	F	F	F	F	F	F	F
		1															16
	2 nd WORD	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Figure 1-12. Complex Format

MEMORY ADDRESSING

A memory reference instruction can use several techniques for addressing memory: direct addressing, indexing, and indirect addressing. (See Figure 1-13). Indexing and indirect addressing may be specified in the same instruction, and indexing may be pre- or post-indirect addressing. Multi-level indirect addressing is provided.

Direct Addressing

The memory of the Honeywell 316/516 is considered to be divided into sectors of 512 words each (i.e., a 4096-word computer will have eight sectors). Any word in a sector can be addressed with nine bits ($2^9 = 512$). The address portion of a memory reference instruction (bits 8 to 16) can define a unique word in a sector. Addresses within sectors run from $(000)_8$ to $(777)_8$. The sector bit, bit 7 of the instruction, identifies the sector of the word addressed in accordance with the following rules:

Sector Bit = 0 The address is in sector 0 (octal address 0000 - 00777).

Sector Bit = 1 The address is in the same sector as the instruction being executed.

For example, assume an ADD instruction having an address of 444_8 is in location $(02100)_8$, or sector 2 word 100. If the sector bit in the instruction is 0, the instruction references word 444_8 in sector 0, or $(00444)_8$. If the sector bit is 1, then the instruction references word 444_8 in sector 2, or $(02444)_8$, because the instruction itself is in sector 2.

A single instruction can thus directly address 1024 words, half of which are in sector 0 and half of which are determined by the location of the instruction. Figure 1-13 represents the memory that can be directly addressed by an instruction in sector 2 and an instruction in sector 6.

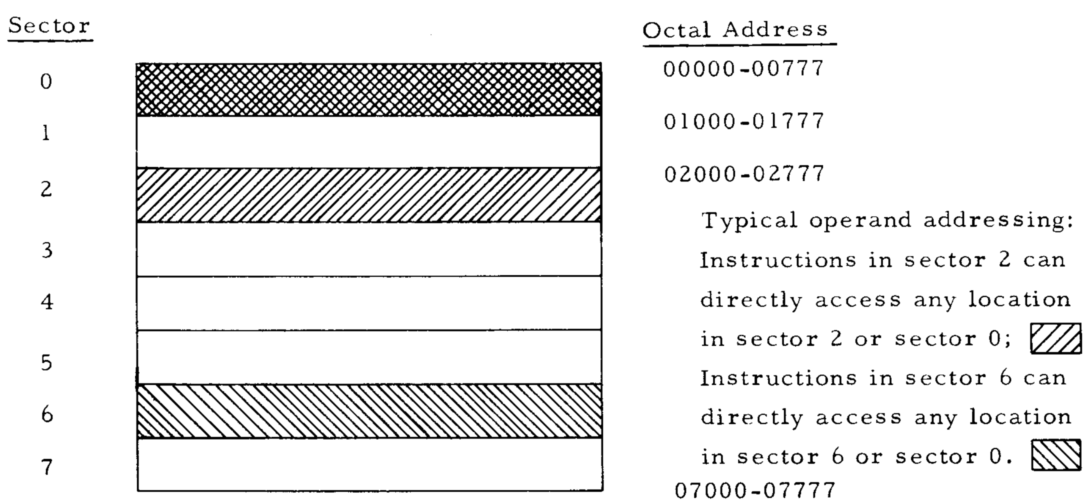


Figure 1-13. Memory Sectors in 4096-Word Honeywell 316/516

Indirect Addressing

If bit 1 of a memory reference instruction is set, indirect addressing takes place. When indirect addressing is specified, the effective address of the operand is assumed to be the content of the location specified by the direct address. The format of the indirect address location is shown in Figure 1-14.

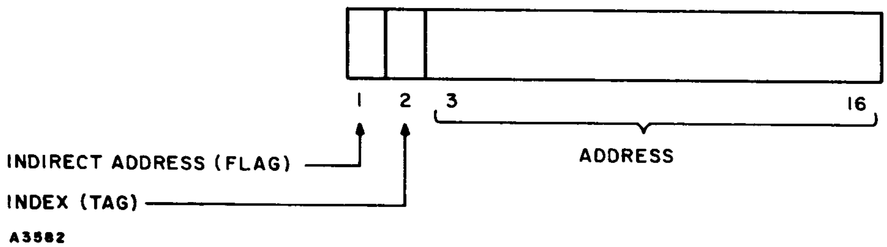


Figure 1-14. Indirect Address Format

To illustrate indirect addressing, consider that an ADD command in sector 2 is flagged for indirect addressing (this is specified in DAP by placing an asterisk after the op code).

ADD* 444_8

Location 444_8 contains

$(06231)_8$

The effective address would then be $(06231)_8$, which is in sector 6. The content of location 06231_8 would be added to the A-register.

Since the address field in the indirect address location is 14 bits, up to 16K of memory can be addressed in this mode. Indirect addressing adds a cycle to the execution time of an instruction.

Multi-Level Indirect Addressing

Bit 1 of the indirect address word also contains a flag bit. If this is set, another level of indirect addressing occurs. This chaining of indirect addressing continues until an indirect address word is reached whose flag bit is zero. Each level of indirect addressing adds a cycle to instruction execution time.

NOTE

With the memory lockout option, instructions executed in the restricted mode cause an interrupt if more than eight levels of indirect addressing are attempted.

INDEXING

The index register is a 16-bit hardware register whose contents can be added to the direct address of an instruction to produce a new effective operand address. This action causes no increase in instruction execution time. Indexing is specified by putting a ONE in bit 2 of a memory reference instruction.

If indexing is specified, the value in the index register is added algebraically to the direct address. The index register can contain either a positive or negative (two's complement) value, although negative values are generally used.

For example, if the index register contained -2 $(177776)_8$, and the ADD 444_8 instruction at $(02100)_8$ mentioned in the previous section were executed with both the index and sector bits set, the effective address would be $(02444)_8 + (177776)_8$ or $(02442)_8$. Sector boundaries can be crossed with no increase in instruction execution time.

The index register can be loaded or stored directly by means of the load index (LDX) and store (STX) instructions. In addition, any instruction that addresses memory location 0 addresses the index register. The usual way of incrementing the index register is by an IRS 0 instruction.

Indirect, Pre-Index

Pre-indexing occurs if both the indirect and index bits of an instruction are set. In this case, indexing is applied to the direct address to determine the location of the indirect address.

Indirect, Post-Index

If the indirect bit in an instruction is set, and if the index bit is set in the indirect location as opposed to the instruction itself, indexing is applied to the indirect address to determine the location of the operand. This action is called post-indexing.

ADDRESSING SUMMARY

Figure 1-15 is a flow chart that shows the various phases in developing the effective address of a memory reference instruction. It is for the normal mode only and does not cover the development of addresses in the following cases:

a. Memory lockout is included in the system and the base (J) register is not zero. (See Section II.)

b. The system contains more than 16K of memory and the extend mode is being utilized. (See Section II.)

Dedicated Locations 1-17

Memory locations $(00001)_8$ through $(00017)_8$ are protected in the standard machine against being written into under program control. Information may be read from these locations in the normal manner. However, all instructions which attempt to write in them

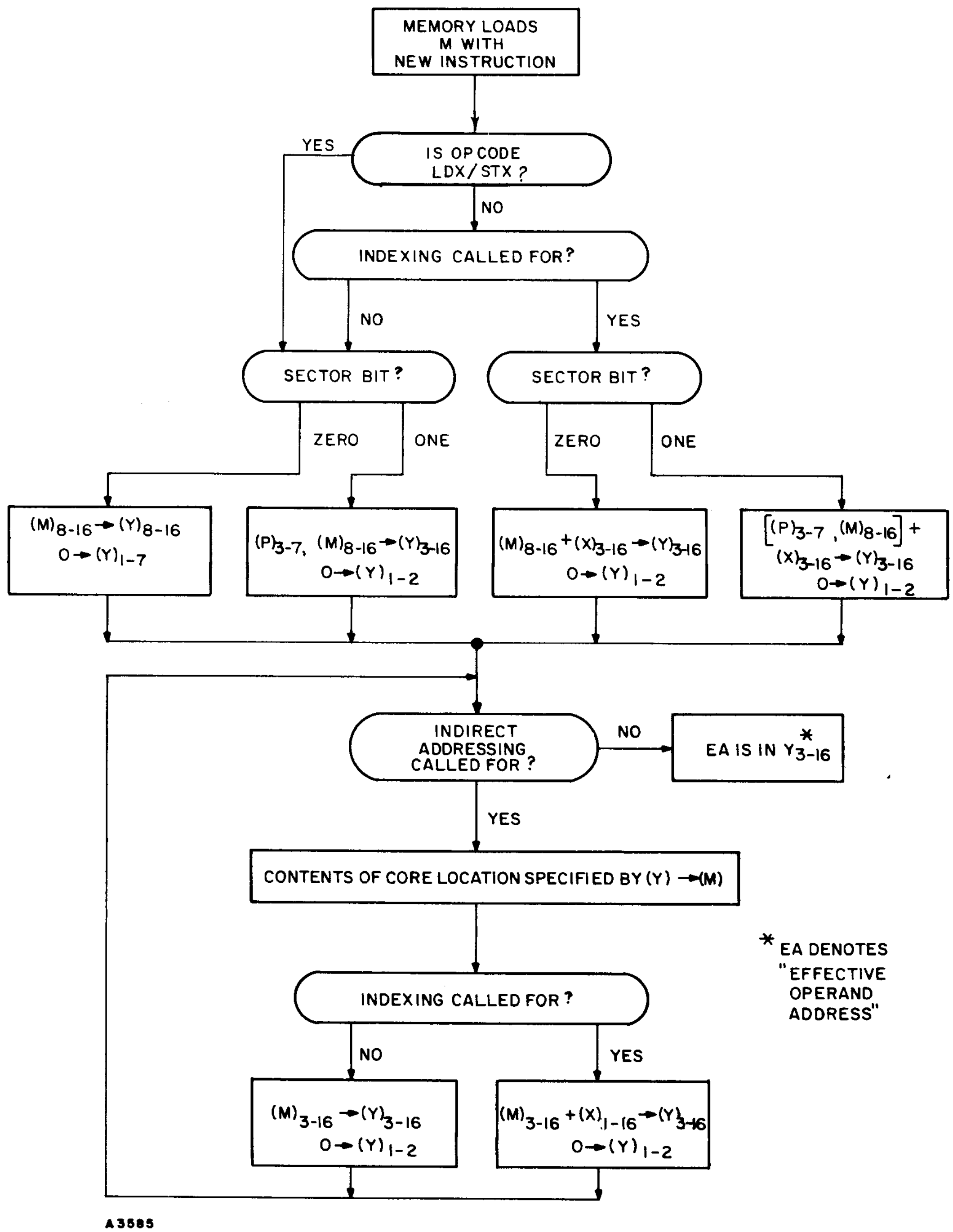


Figure 1-15. Fetch, Indexing, and Indirect Addressing, Logic Flow Diagram

will be aborted. The only way in which these locations may be loaded is through the use of the memory access feature of the console. (See the Honeywell 316/516 Operators Guide.) The locations provide protected storage for the Key-In Loader used with the software system. (See Appendix G.)

Dedicated Location 0 (Index Register)

The hardware index register tracks the dedicated memory location $(00000)_8$ (index register); that is, any modification of location $(00000)_8$ causes the hardware index register to be changed to agree with $(00000)_8$. (For systems with memory lockout, see Section II.)

Instruction Sequences

Programs are executed sequentially with the contents of the program counter (P-register) being incremented by one upon the execution of each instruction. Certain instructions (SKIPS, COMPARE, I/O) conditionally increment the program counter by an additional one or two, thereby causing a skip. Others (JUMP, JUMP-STORE) unconditionally load the program counter with the effective address, thereby causing a branch in the program.

Breaks

Certain operations may occur between instructions or between cycles of instructions without effecting the contents of the program counter. When the operations are complete, the program resumes. These actions are called "breaks," and include such operations as DMA or DMC I/O cycles, incrementation of the real-time clock, and memory increment breaks.

Interrupts

An interrupt is different from a break in that an action occurring independently of a program can cause the contents of the program counter to be automatically changed, thereby changing the sequence of instruction execution. Interrupts have unique memory locations dedicated to them whose contents are interpreted as an indirect address. The action of an interrupt causes the program to branch to the location whose address is stored in the dedicated location.

Interrupts are caused by:

a. I/O interrupts
b. Power Failure Interrupt
c. Memory Lockout Interrupts
d. Additional Interrupts
e. Start Button

Memory Access Priority Structure

The various functions that the computer performs are executed in a priority sequence if two or more functions are trying to simultaneously access memory. The following table shows the relative priorities between the program and breaks and interrupts. Details on the latter are explained in the following chapters.

Table 1-1.
Honeywell 316/516 Computer Access-to-Memory Priority Structure

Relative Priority Level	Option/Function
1	Direct Memory Access Break (DMA) DDP-516-21
2	Direct Multiplex Control Break (DMC)H316-20, 21, DDP-516-20
3	Power Failure Interrupt (PFI), Standard
4	Real-Time Clock Break Honeywell 316/516-12
5	Memory Lockout Violation Interrupt, DDP-516-08
6	Standard Interrupt Standard
7	Memory Increment Break Honeywell 316/516-26
8	Priority Interrupt Honeywell 316/516-25
9	Central Processing Unit (CPU)

SECTION II
INSTRUCTION REPERTOIRE

The instructions which comprise the standard Honeywell 316/516 instruction repertoire are described in this section. Mnemonics and symbols used in the instruction descriptions are listed in Table 2-1. A thorough knowledge of the data presented in Table 2-1 is necessary to understand the instruction descriptions.

Tables 2-2 through 2-9 list all standard instructions. Each instruction is identified by its assigned three-letter mnemonic, type symbol, and octal op code. Definitions, descriptions, and timing data for each instruction are also included in these tables. (See Section I for instruction word formats.)

STANDARD INSTRUCTIONS

The standard instructions in Tables 2-2 through 2-9 are grouped into the following operational categories:

a. Load and Store
b. Arithmetic
c. Logical
d. Shift
e. Input/Output
f. Control
g. Half-Word

Arithmetic instructions which provide overflow detection are indicated by the designation Overflow Status → (C). If overflow occurs on a particular instruction, the C-bit is set to a one. If overflow does not occur, the C-bit is reset to a zero. Thus, after each arithmetic instruction, the contents of the C-bit indicate whether or not overflow occurred on that instruction.

STANDARD INTERRUPT

The Honeywell 316/516 has an interrupt system to which all devices are connected by means of the priority interrupt line (PIL) of the I/O bus. For a device to cause an interrupt, the following conditions must be met:

a. The device must be ready.
b. The interrupt mask flip-flop must be set. (See SMK instruction.)
c. System interrupt must be enabled by an ENB instruction.

All interrupts are stored until they are serviced. An interrupt request is removed by the action of an INA or OTA command, resetting the ready status.

Table 2-1.
Glossary of Symbols

Symbol	Definition
EA	Effective operand address; the address from which the operand is obtained. This is determined only after all selection of sectors, indexing, and indirect addressing have been performed.
n	Specified number of shifts to be performed.
N	Two's complement of the number of shifts to be performed.
ADB	Address Bus
INB	Input Bus
OTB	Output Bus
EXTMD	Extended Mode Indicator - associated with Extended Addressing - Honeywell 516-05, 06
DP Mode	Double Precision Mode associated with Honeywell 316/516-11
A	A-Register (16 bits)
P	Program Counter (16 bits) -
B	B-Register (16 bits)
E	E-Register (16 bits)
X	Index Register (16 bits)
M	M-Register (16 bits)
C	C-bit (1 bit)
$\longrightarrow$	Replaces
$\leftrightarrows$	Is exchanged with
$\overline{\quad}\!\downarrow$	Is discarded
$\wedge$	Logical AND
$\vee$	Logical OR
$\veebar$	Exclusive OR
+	Algebraic Addition
( )	Contents of a hardware register (e. g., (A) = contents of A-Register)
[ ]	Contents of core location specified by (e. g. [EA] = contents of core location specified by EA)
T	Tag Bit (bit 2 of instruction word)
MR	Memory Reference Instruction
G	Generic Instruction
SH	Shift Instruction
IO	Input-Output Instruction

Table 2-2.
Load and Store Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (µs) DDP-516	Time (µs) H316
CRA	G	140040	Clear A	$0 \rightarrow (A)$	1	0.96	1.6
IAB	G	000201	Interchange A and B	$(A) \rightleftarrows (B)$	1	0.96	1.6
IMA	MR	13	Interchange Memory and A	$(A) \rightleftarrows [EA]$	3	2.88	4.8
INK	G	000043	Input Keys	$(C) \rightarrow (A)_1$ $(DP\ Mode) \rightarrow (A)_2$ $(PMI) \rightarrow (A)_3$ $0 \rightarrow (A)_{4-11}$ $Shift\ Count \rightarrow (A)_{12-16}$	1	0.96	1.6
LDA	MR	02	Load A	$[EA] \rightarrow (A)$	2	1.92	3.2
LDX	MR	15 T = 1	Load X	$[EA] \rightarrow (X)$ $[EA] \rightarrow [00000]$ NOTE This instruction cannot be indexed. However, if indirect addressing is called for, the indirect address can be indexed in the usual manner.	3	2.88	4.8
OTK	G	171020	Output Keys	$(A)_1 \rightarrow (C)$ $(A)_2 \rightarrow (DP\ Mode)$ $(A)_3 \rightarrow (EXTMD)$ $(A)_{12-16} \rightarrow Shift\ Count$	2	1.92	1.6
STA	MR	04	Store A	$(A) \rightarrow [EA]$	2	1.92	3.2
STX	MR	15 T = 0	Store X	$(X) \rightarrow [EA]$ NOTE This instruction cannot be indexed. However, if indirect addressing is called for, the indirect address can be indexed in the usual manner.	2	1.92	3.2

Table 2-3.
Arithmetic Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs) DDP-516	H316
ACA	G	141216	Add C to A	$(A) + (C) \rightarrow (A)$ Overflow status $\rightarrow$ (C)	1	0.96	1.6
ADD	MR	06	Add	$(A) + [EA] \rightarrow (A)$ Overflow status $\rightarrow$(C)	2	1.92	3.2
AOA	G	141206	Add One to A	$(A) + 1 \rightarrow (A)$ Overflow status $\rightarrow$ (C)	1	0.96	1.6
SUB	MR	07	Subtract	$(A) - [EA] \rightarrow (A)$ Overflow status $\rightarrow$ C	2	1.92	3.2
TCA	G	141407	Two's Complement A	$(\overline{A}) + 1 \rightarrow (A)$	1.5	1.44	2.4

Table 2-4.
Logical Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs) DDP-516	
ANA	MR	03	AND to A	$(A) \wedge [EA] \rightarrow (A)$ EXAMPLE: (A) 0 1 0 1 [EA] 0 0 1 1 RESULT IN A 0 0 0 1	2	1.92	3.2
CSA	G	140320	Copy Sign and Set Sign Plus	$(A)_1 \rightarrow (C)$ $0 \rightarrow (A)_1$	1	0.96	1.6
CHS	G	140024	Complement A Sign	$\overline{(A)}_1 \rightarrow (A)_1$	1	0.96	1.6
CMA	G	140401	Complement A	$\overline{(A)} \rightarrow (A)$(ONEs complement)	1	0.96	1.6
ERA	MR	05	Exclusive OR to A	$(A) \veebar [EA] \rightarrow (A)$ EXAMPLE: (A) 0 0 1 1 [EA] 0 1 0 1 RESULT IN A 0 1 1 0	2	1.92	3.2
SSM	G	140500	Set Sign Minus	$1 \rightarrow (A)_1$	1	0.96	1.6
SSP	G	140100	Set Sign Plus	$0 \rightarrow (A)_1$	1	0.96	1.6

Table 2-5.
Shift Instruction Repertoire

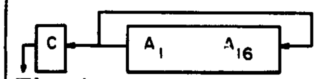
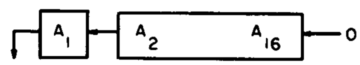
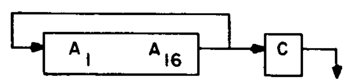
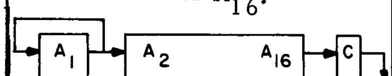
Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)
ALR	SH	0416	Logical Left Rotate	 The A register is shifted left, end-around (n) positions. A_1 is shifted out to A_{16} and the C bit. The C bit takes the state of the last bit shifted into A_{16}.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
ALS	SH	0415	Arithmetic Left Shift	 Overflow status → (C) The A register is shifted left (n) positions. If shifting causes a change in the sign of A at any time during the instruction, the C bit is set. If the sign is not changed, the C bit is reset. After 16 or more shifts, the A register contains zero.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
ARR	SH	0406	Logical Right Rotate	 The A register is shifted right, end around (n) positions. Bits shifted out of A_{16} enter A_1 and the C bit. The C bit takes the state of the last bit shifted out of A_{16}.	1+n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
ARS	SH	0405	Arithmetic Right Shift	 The A register is shifted right (n) position. The sign bit (A_1) does not change; it is shifted into vacated positions of the register. Bits shifted out of A_{16} enter the C bit. The C bit takes the state of the last bit shifted out of A_{16}. If 15 or more shifts are specified, all bits of the A register are the same as the sign bit.	1+n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)

Table 2-5. (Cont)
Shift Instruction Repertoire

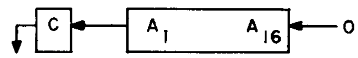
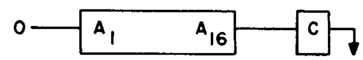
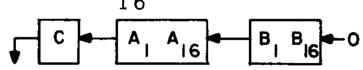
Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)
LGL	SH	0414	Logical Left Shift	 The A register is shifted left (n) positions. Zeros are filled in vacated bit positions. A_1 is shifted to the C bit. Bits shifted out of C are discarded. After 16 or more shifts, the A register contains zero. The C bit takes the state of the last bit shifted out of A_1.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
LGR	SH	0404	Logical Right Shift	 The A register is shifted right (n) positions. Zeros fill in vacated bit positions. A_{16} is shifted to the C bit. Bits shifted out of C are discarded. After 16 or more shifts, the A register contains zero. The C bit takes the state of the last bit shifted out of A_{16}.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
LLL	SH	0410	Long Left Logical Shift	 The A and B registers are treated as a single 32-bit register (A being the most significant) and shifted left n positions. Zeros are shifted into vacated positions of B. Bits are shifted from B_1 to A_{16}. Each bit shifted out of A_1 enters the C bit. Bits shifted out of the C bit are discarded. If 32 or more shifts are specified, the A and B registers contain zero. The C bit takes the state of the last bit shifted out of A_1.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)

Table 2-5. (Cont)
Shift Instruction Repertoire

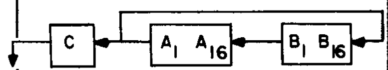
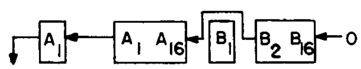
Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)
LLR	SH	1412	Long Left Rotate	 The A and B registers are treated as a single 32-bit register and shifted left, end-around, (n) positions. Bits shifted out of B_1 enter A_{16}; bits shifted out of A_1 enter B_{16} and the C bit. Bits shifted out of C bit are discarded. The C bit takes the state of the last bit shifted into B16.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
LLS	SH	0411	Long Arithmetic Left Shift	 Overflow Status (C) The A and B registers are treated as a single 31-bit register (B_1 is not changed) and shifted left n positions. Zeros are shifted into vacated positions through B_{16}. Bits shifted out of B_2 enter A_{16}. If at any time during the instruction the sign of the A register $(A)_1$ is changed, the C bit is set. If at the end of the instruction the sign has not been changed, the C bit is reset. If 31 or more shifts are specified, the A and B registers contain zero (except for B_1, which is unchanged).	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)

Table 2-5. (Cont)
Shift Instruction Repertoire

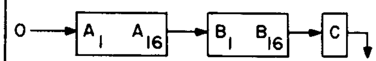
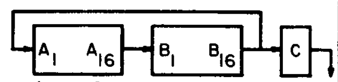
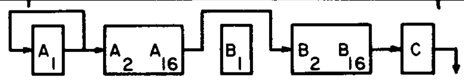
Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)
LRL	SH	0400	Long Right Logical Shift	 The A and B registers are treated as a single 32-bit register (A being the most significant) and shifted right n positions. Bits shifted out of A_{16} enter B_1. Bits shifted out of B_{16} enter the C bit. Bits shifted out of C bit are discarded. Zeros are shifted into vacated positions through A_1. The C bit takes the state of the last bit shifted out of B_{16}. If 32 or more shifts are specified, the A and B registers contain zero.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)
LRR	SH	0402	Long Right Rotate	 The A and B registers are treated as a single 32-bit register (A being the most significant) and shifted right, end-around (n) positions. Bits shifted out of A_{16} enter B_1. Bits shifted out of B_{16} enter A_1 and the C bit. Bits shifted out of C are discarded. The C bit takes the state of the last bit shifted into A_1.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)

Table 2-5. (Cont)
Shift Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (µs)
LRS	SH	0401	Long Arithmetic Right Shift	 The A and B registers are treated as a single 31-bit register (B_1 is not changed) and shifted right (n) positions. The sign bit, A_1, is not changed; it is propagated into vacated positions of the register. Bits shifted out of A_{16} enter B_2. Bits shifted out of B_{16} enter the C bit. (Bits shifted out of the C bit are discarded.) After 30 or more shifts, both registers are filled with the sign of the A register, except for B_1 which is unchanged. The C bit takes the state of the last bit shifted out of B_{16}.	1 + n/2	0.96 + 0.48n (DDP-516) 1.6 + 0.8n (H316)

Note: The C bit is always cleared before a shift instruction is executed.

Table 2-6.
Half-Word Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)	
						DDP-516	H316
CAL	G	141050	Clear A, Left Half	$0 \rightarrow (A_{1-8})$ (A_{9-16}) are unchanged	1	0.96	1.6
CAR	G	141044	Clear A, Right Half	$0 \rightarrow (A_{9-16})$ (A_{1-8}) are unchanged	1	0.96	1.6
ICA	G	141340	Interchange Characters in A	$(A_{1-8}) \leftrightarrows (A_{9-16})$ A_1 is interchanged with A_9, A_2 with A_{10}, etc.	1	0.96	1.6
ICL	G	141140	Interchange and Clear Left Half of A	$(A_{1-8}) \rightarrow (A_{9-16})$ $0 \rightarrow (A_{1-8})$ Bits 9-16 of A are replaced with bits 1-8; bits 1-8 are cleared.	1	0.96	1.6
ICR	G	141240	Interchange and Clear Right Half of A	$(A_{9-16}) \rightarrow (A_{1-8})$ $0 \rightarrow (A_{9-16})$ Bits 1-8 of A are replaced with bits 9-16; bits 9-16 are cleared.	1	0.96	1.6

Table 2-7.
Control Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs) DDP-516	Time (μs) H316
CAS	MR	11	Compare	Algebraically compare (A) and [EA] If (A) > [EA], execute next instruction. If (A) = [EA], skip next instruction. If (A) < [EA], skip two instructions.	3	2.88	4.8
ENB	G	000401	Enable Program Interrupt	Set machine status to permit interrupt. The permit interrupt status does not take effect until the instruction immediately following ENB is completed. (PI indicator lights.)	1	0.96	1.6
HLT	G	000000	Halt	Sets machine to halt mode. No further instructions or interrupts are serviced until the console START button is pressed, at which time normal execution resumes.			
INH	G	001001	Inhibit Program Interrupt	Resets "permit interrupt status" to prohibit standard or priority interrupts. (PI indicator is extinguished.)	1	0.96	1.6

H316

Table 2-7. (Cont)
Control Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs) DDP-516	Time (μs) H316
IRS	MR	12	Increment, replace and Skip	$[EA] + 1 \rightarrow [EA]$ If $[EA] + 1 = 0$, skip next instruction	3	2.88	4.8
JMP	MR	01	Unconditional Jump	$EA \rightarrow (P)$ Next instruction to be executed is at location EA.	1	0.96	1.6
JST	MR	10	Jump and Store Location	$(P_{3-16}) \rightarrow [EA_{3-16}]$ $[EA_{1,2}]$ not changed $EA_{3-16} + 1 \rightarrow (P_{3-16})$	3	2.88	4.8
NOP	G	101000	No Operation	Performs no operation. Computer proceeds to next instruction.	1	0.96	1.6
RCB	G	140200	Reset C Bit	$0 \rightarrow (C)$	1	0.96	1.6
SCB	G	140600	Set C Bit	$1 \rightarrow (C)$	1	0.96	1.6
SKP	G	100000	Unconditional Skip	Skip next instruction	1	0.96	1.6
SLN	G	101100	Skip if (A_{16}) One	If $(A_{16}) = 1$: skip next instruction	1	0.96	1.6
SLZ	G	100100	Skip if (A_{16}) Zero	If $(A_{16}) = 0$: skip next instruction	1	0.96	1.6
SMI	G	101400	Skip if A Minus	If $(A_1) = 1$: skip next instruction	1	0.96	1.6
SNZ	G	101040	Skip if A Not Zero	If $(A) \neq 0$: skip next instruction	1	0.96	1.6
SPL	G	100400	Skip if A Plus	If $(A_1) = 0$: skip next instruction	1	0.96	1.6
SR1	G	100020	Skip if Sense Switch 1 is Reset	If Sense Switch 1 is OFF: skip next instruction	1	0.96	1.6
SR2	G	100010	Skip if Sense Switch 2 is Reset	If Sense Switch 2 is OFF: skip next instruction	1	0.96	1.6
SR3	G	100004	Skip if Sense Switch 3 is Reset	If Sense Switch 3 is OFF: skip next instruction	1	0.96	1.6

Table 2-7. (Cont)
Control Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs) DDP-516	Time (μs) H316
SR4	G	100002	Skip if Sense Switch 4 is Reset	If Sense Switch 4 is OFF: skip next instruction	1	0.96	1.6
SRC	G	100001	Skip if C Reset	If (C) = 0: skip next instruction	1	0.96	1.6
SS1	G	101020	Skip if Sense Switch 1 is Set	If Sense Switch 1 is ON: skip next instruction	1	0.96	1.6
SS2	G	101010	Skip if Sense Switch 2 is Set	If Sense Switch 2 is ON: skip next instruction	1	0.96	1.6
SS3	G	101004	Skip if Sense Switch 3 is Set	If Sense Switch 3 is ON: skip next instruction	1	0.96	1.6
SS4	G	101002	Skip if Sense Switch 4 is Set	If Sense Switch 4 is ON: skip next instruction	1	0.96	1.6
SSC	G	101001	Skip if C Set	If (C) = 1: skip next instruction	1	0.96	1.6
SSR	G	100036	Skip if No Sense Switch Set	If no Sense Switches are ON: skip next instruction	1	0.96	1.6
SSS	G	101036	Skip if Any Sense Switch Set	If any Sense Switch is ON: skip next instruction	1	0.96	1.6
SZE	G	100040	Skip if A Zero	If (A) = 0: skip next instruction	1	0.96	1.6

Table 2-8.
Input/Output Instruction Repertoire

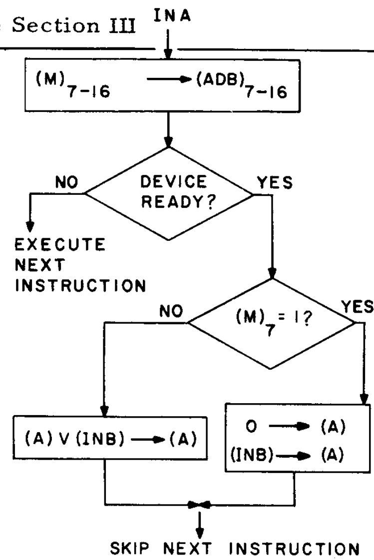
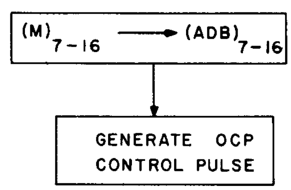
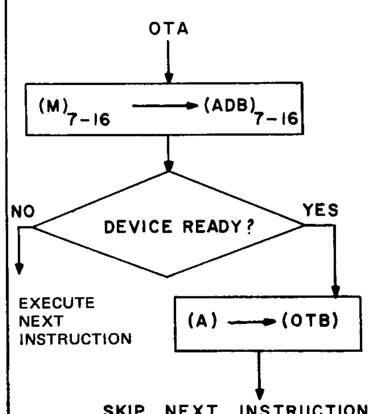
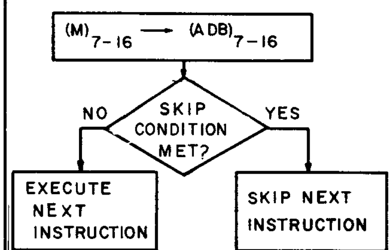
Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)	
						DDP-516	H316
Input-Output		For I/O Discussion see Section III					
INA	IO	54 For INA codes, see Appen-dices E and F.	Input to A		2	1.92	3.2
OCP	IO	14 For OCP codes, see Appen-dices E and F.	Output Control Pulse		2	1.92	3.2
OTA	IO	74 For OTA codes, see Appen-dices E and F.	Output from A		2	1.92	3.2

Table 2-8. (Cont)
Input/Output Instruction Repertoire

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs) DDP-516	Time (μs) H316
SMK	IO	74 For SMK codes, see Appendices E and F.	Set Mask (Special OTA)	(A) → (OTB) Generate SMK pulse to transfer output bus to external device mask flip-flops. This instruction does not skip.	2	1.92	3.2
SKS	IO	34 For SKS codes, see Appendices E and F.	Skip if Ready Line Set		2	1.92	3.2

I/O interrupts are serviced at the end of an instruction unless they are delayed by higher priority systems such as DMC. (See Section I.) The action of the standard interrupt is to cause a forced indirect jump-store through location 63_8 (JST* 63). This takes three cycles and also forces an inhibit interrupt instruction (INH).

The interrupt subroutine (whose starting address is stored in 63_8) can then reset the mask flip-flop for lower priority devices, leaving those of higher priority still set and re-enable interrupt. Upon exiting the interrupt subroutine, it can then reset the lower priority mask.

This allows the Honeywell 316/516 to have a very flexible priority interrupt system with high priority interrupts interrupting lower priority interrupts. Sixteen levels of interrupt can be controlled in one instruction by means of the SMK command.

Location and Mask Bits and Device Address

Table 2-9.
Standard Interrupt Mask Assignments

OTB Bit No.	Device	OTB Bit No.	Device
1	Mag Tape Control Unit No. 1	9	Paper Tape Reader
2	Mag Tape Control Unit No. 2	10	Paper Tape Punch
3	(Unassigned)	11	ASR-33/35
4	Moving Head Disc File	12	Card Reader
5	I/O Channel No. 1	13	(Unassigned)
6	I/O Channel No. 2	14	Line Printer
7	I/O Channel No. 3	15	Memory Parity (DDP-516)
		15	(Unassigned) (H316)
8	Small Mass Store	16	Real Time Clock

OPTIONAL INSTRUCTIONS

Extended Addressing (DDP-516)

Memory expansion above 16K in the DDP-516 is handled by the introduction of the "extend" mode. The program counter bit (P02) provides the fifteenth bit of the 32K address field and conditions bit 2 of the Y-register (Y02) when the sector being addressed is not zero.

The extend mode changes the interpretation of the index bit of the indirect address word, which becomes part of a 15-bit indirect address. Only one level of indexing is possible in the extend mode. It is specified by bit 2 of the instruction word and is always the final operation in generating the effective operand address. (See Figure 2-1 for flow-grams illustrating the operation of a system with extended addressing.)

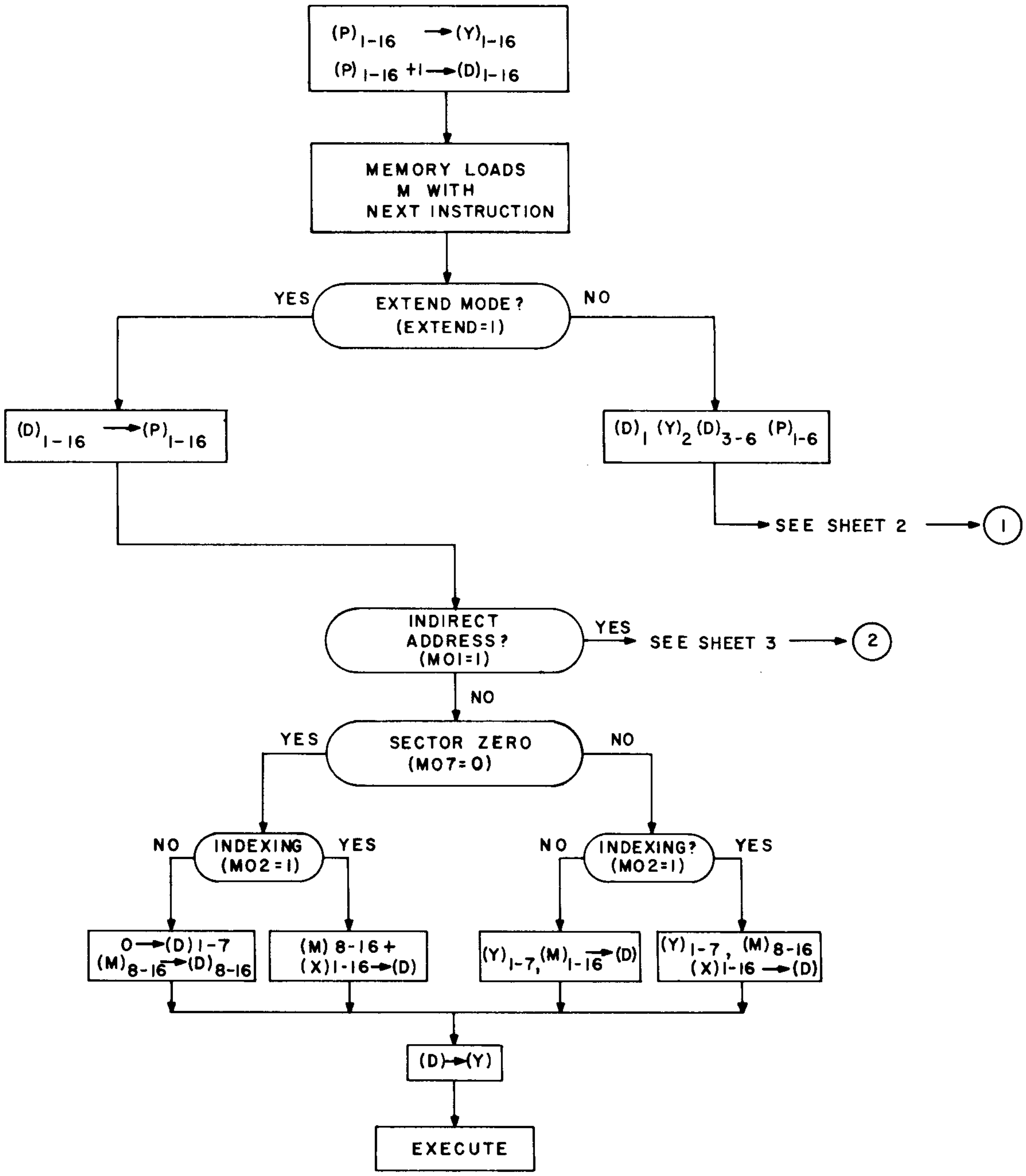


Figure 2-1. Operation of a System with Extended Addressing, Flow Diagram (Sheet 1 of 3)

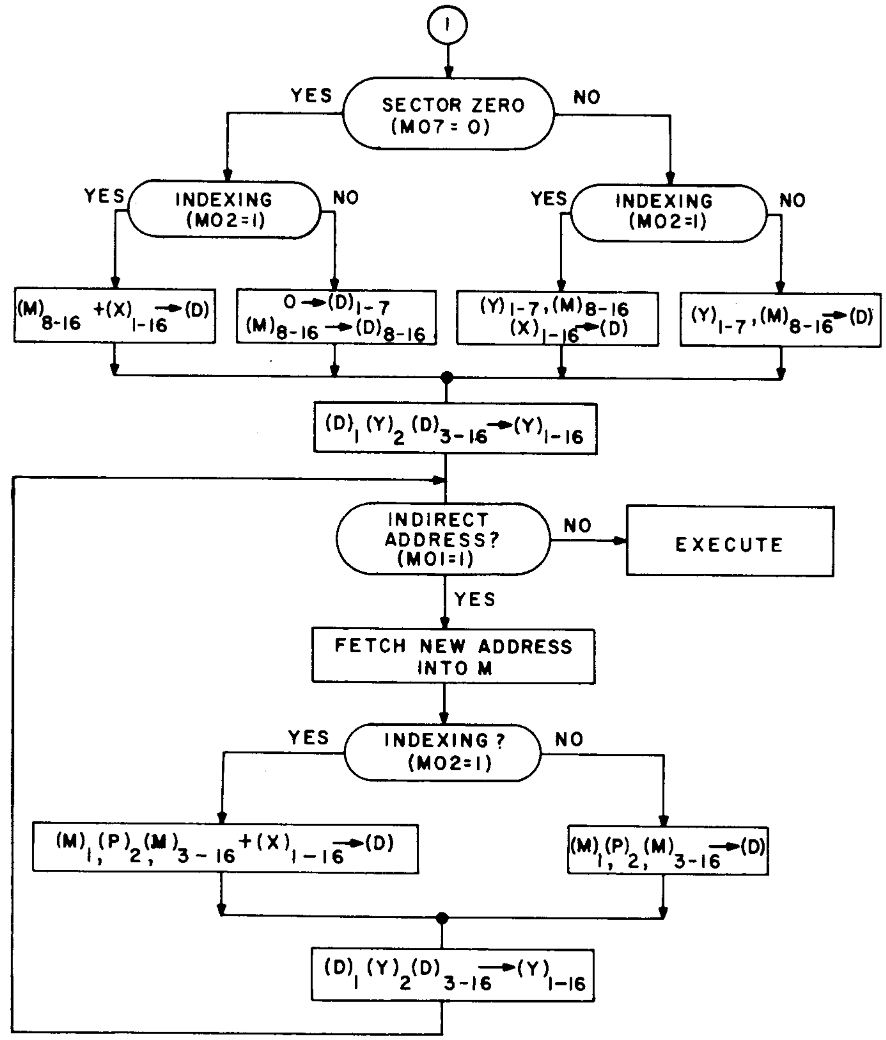


Figure 2-1. Operation of a System with Extended Addressing, Flow Diagram (Sheet 2 of 3)

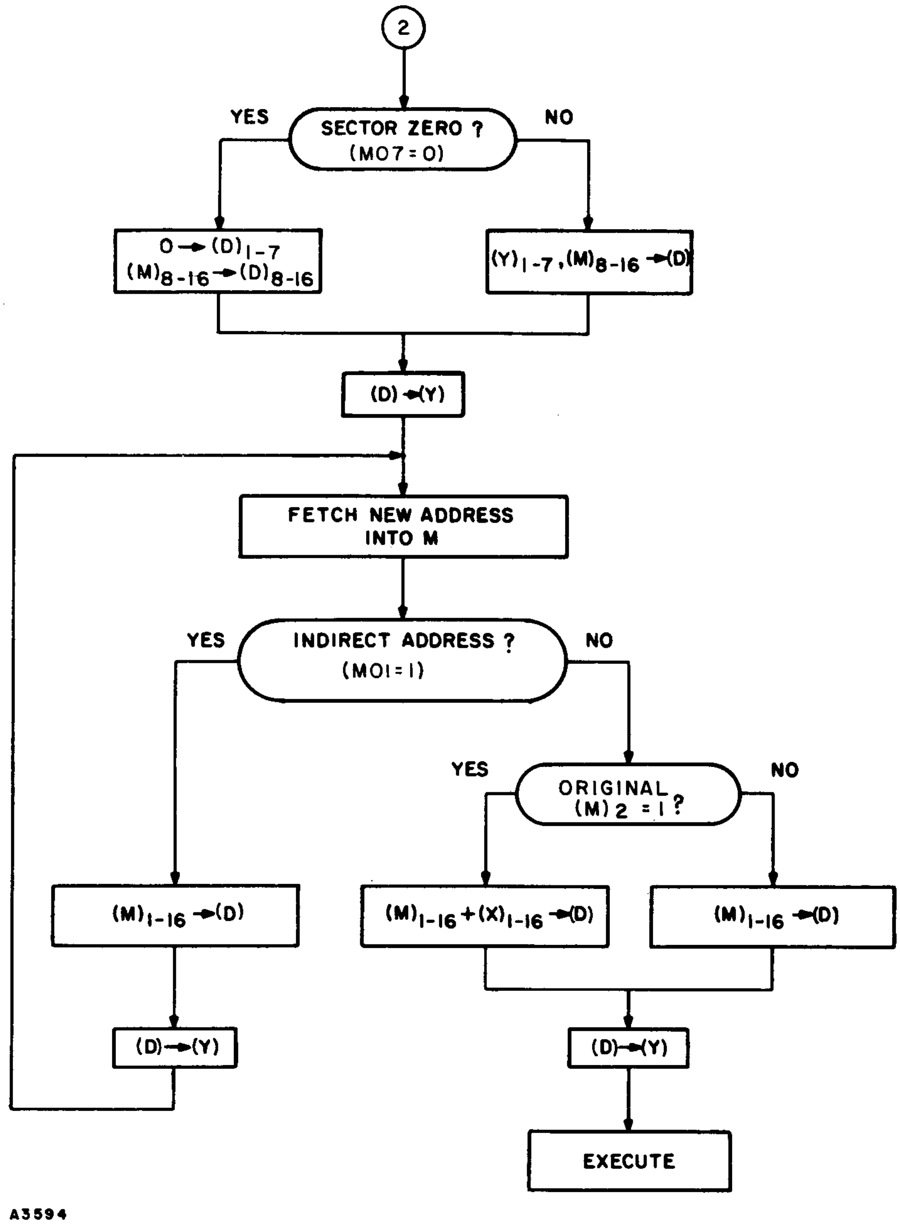


Figure 2-1. Operation of a System with Extended Addressing, Flow Diagram (Sheet 3 of 3)

Operation

The extended mode indicator (EXTMD) is set or reset by the generic instructions EXA or DXA, respectively, and by an OTK, set if $(A)_3$ is a one, reset if a zero. It is also set by the occurrence of a program interrupt. An indication that the computer is in an extend mode may be displayed at the control panel. (See the Honeywell 316/516 Operators Guide.) A previous mode indicator (PMI) is added to the mainframe to save the mode in which the program was operating when a program interrupt occurred.

The PMI is set if the CPU is in the extend mode when a priority interrupt occurs. It is reset if the CPU is not in the extend mode when a priority interrupt occurs and when the control panel MASTER CLEAR pushbutton is depressed.

Instruction Complement

Table 2-10 contains a list of instructions required for extended addressing.

Table 2-10.
Extended Addressing Instructions

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)
EXA*	G	000013	Enable Extended Addressing.	Places computer in extend mode by setting EXTMD (Extend Mode Indicator).	1	0.96
DXA*	G	000011	Disable Extended Addressing.	Restores computer to normal mode. Mode change not effective until after a JMP (01) has been executed to enable proper return from an interrupt subroutine. Any number of non-JMP instructions may be included between the DXA and the first JMP instruction.	1	0.96

*See INK, OTK instructions Table 2-2.
EXTMD will reset on JMP (01) after disabling OTK. Same as DXA.

NOTE

The extend mode alters the JST instruction to allow it to store a 15-bit program counter. Bit 1 of the memory location specified by the effective operand address is left unchanged.

MEMORY PARITY (DDP-516)

The memory parity option enables generation of parity on all memory write cycles and checking of parity on all memory read cycles. An exception exists in that parity is not checked during a console memory read operation. The memory parity error flip-flop in the computer is set when a memory parity error occurs and can be tested and reset under program control. It can also be displayed on the computer control panel. The MASTER CLEAR pushbutton switch on the control panel resets the parity error flip-flop. When the parity error flip-flop is set, an interrupt is generated on the standard interrupt line. This interrupt can be masked on or off by the parity error mask bit (bit 15).

Instruction Complement

The instructions added when this option is included in a system are listed in Table 2-11.

Table 2-11.
Memory Parity Instructions

Mnemonic	Type	Instruction Word	Definition	Description	No. of Cycles	Time (μs)
RMP	G	000021	Reset Memory Parity Error.	Resets memory parity error flip-flop.	1	0.96
SPS	G	101200	Skip on Memory Parity Error.	Skips next instruction if parity error flip-flop is set.	1	0.96
SPN	G	100200	Skip on No Memory Parity Error.	Skips next instruction if parity error flip-flop is reset.	1	0.96
SMK '0020	I/O	170020	Set Mask.	$(A)_{15} \rightarrow$ Parity Interrupt Mask 1. $(A_{15}) = 1$, enable interrupt 2. $(A_{15}) = 0$, inhibit interrupt	2	1.92

MEMORY LOCKOUT OPTION (DDP-516-08)

The memory lockout option facilitates the time-shared execution of various programs. The option provides base sector relocation to facilitate desectorization of more than one program. It also equips the CPU with a mode of operation called the "restricted mode" which enables unverified programs to be time-shared with other programs.

Base Sector Relocation

The memory lockout option provides for the relocation of the base sector insofar as the latter term applies to address information. The option includes a 6-bit base sector relocation register (J, non-readable) used to identify the physical sector currently assigned as the base sector. When the sector bit, bit 7 of the instruction word, is a one, the address (bits 8-16) is in the same sector as the instruction being performed. This represents no change from the basic machine. When the sector bit is a zero, the memory lockout option forces the address to be in the sector specified by the base sector relocation register. Figure 2-2 contains a flow chart that shows when base sector relocation takes place relative to indexing extended addressing and indirect addressing. If physical sector zero is called for as a result of indexing, it is not relocated.

Base sector relocation does not affect memory references caused by breaks or program interrupts. The base sector relocation register can be changed by an SMK '1320 instruction. (See Table 2-12.) Any program interrupt, as well as MASTER CLEAR, clears this register.

Memory location $(00000)_8$ and the hardware index register do not agree after the relocation of the base sector. Before any indexing is attempted, the relocated $(00000)_8$ should be modified (STA or LDX) to get the hardware register in step with the relocated $(00000)_8$. This operation must be repeated when the base sector is returned to sector zero.

Restricted Mode

There are two modes of operation associated with the memory lockout option: restricted and normal modes. The restricted mode has the following properties:

a. Instructions which normally write into memory locations can be "locked out" of protected memory sectors. These instructions are STA, LDX, STX, IMA, IRS, and JST.

b. Certain instructions are considered illegal and cannot be performed. They are OCP, SKS, OTA, INA, SMK, HLT, and INH.

c. Indirect addressing is limited to eight levels.

If executed in the restricted mode, SMK, OTA, INA, OCP, and SKS instruction cause a memory lockout violation and request an interrupt (location $00062)_8$, which occurs at the end of the violating instruction. OCP and SMK are treated as NOPs, SKS is unchanged, and an OTA or INA is treated as an SKS. If the device was ready (skip condition), the INA sets the A register to all ones. For the SMK, OCP and the non-skip (device not ready) case of OTA, INA, SKS, the interrupt JST stores the address of the violating instruction +1. For the skip cases of OTA, INA, and SKS, the interrupt JST stores the address of the violating instruction +2.

If attempting to alter a location in a protected sector, while in the restricted mode, STA, STX, LDX, IMA, IRS, or JST causes a memory lockout violation. The violating STA, STX or LDX is treated as an NOP, IMA as an LDA, IRS as an IRS except that the protected memory location is not modified and the JST as an unconditional JMP to EA + 1. The memory lockout violation interrupt is strobed into the interrupt priority network during the next clock cycle after the completion of the violating instruction. There are three cases for this interrupt:

a. If the next clock cycle is a DMA, DMC, RTC, or MI cycle, the memory lockout violation interrupt occurs after such a cycle and the interrupt JST stores the address of the violating instruction +1.

b. If the next clock cycle is the next instruction in the program it is processed normally (subject to restricted mode operation) and the memory lockout violation interrupt occurs at the completion of this instruction. The interrupt JST stores the address of the violating instruction + 2.

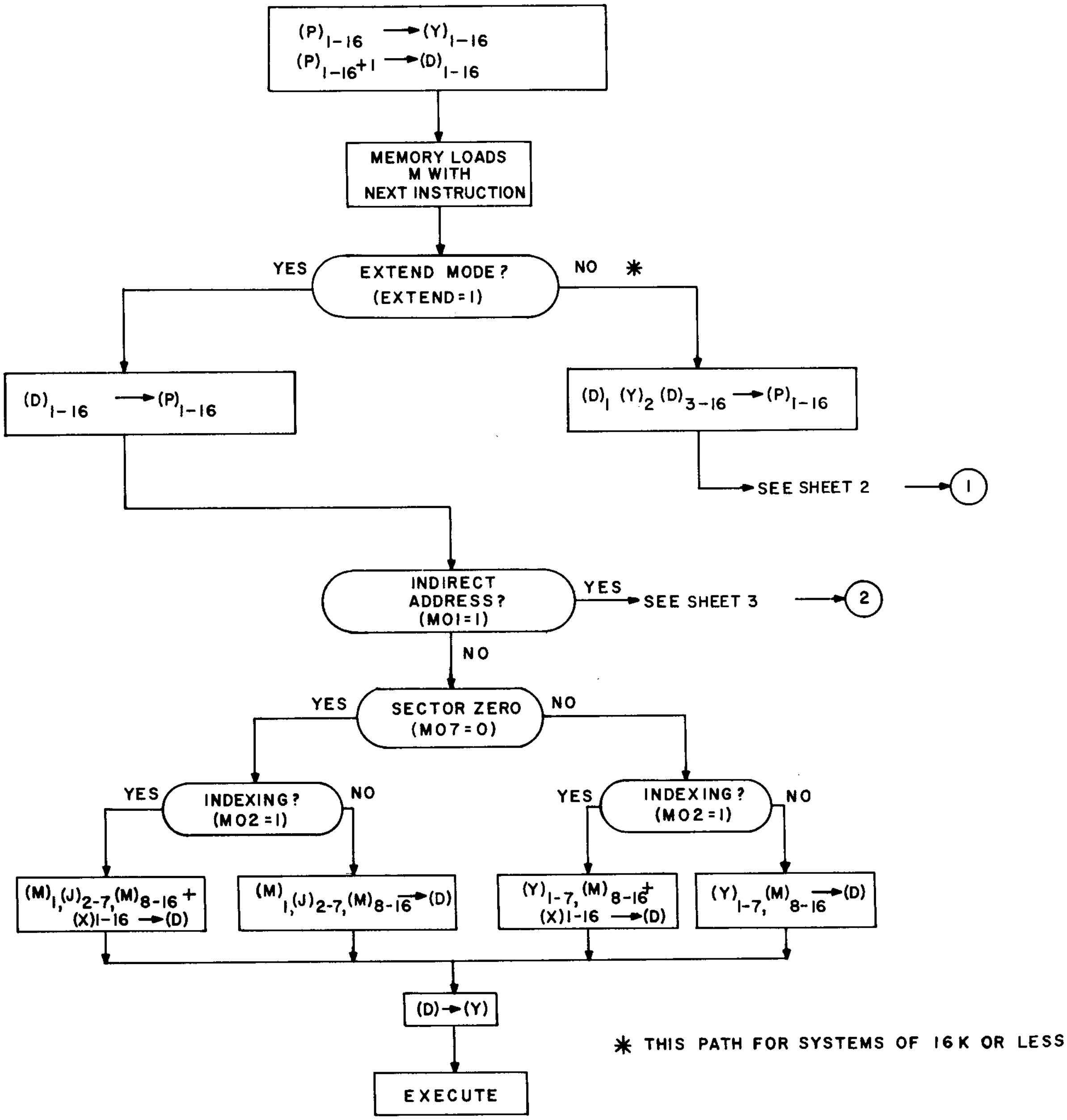


Figure 2-2. Operation of a System with Memory Lockout and up to 32K of Memory (Sheet 1 of 3)

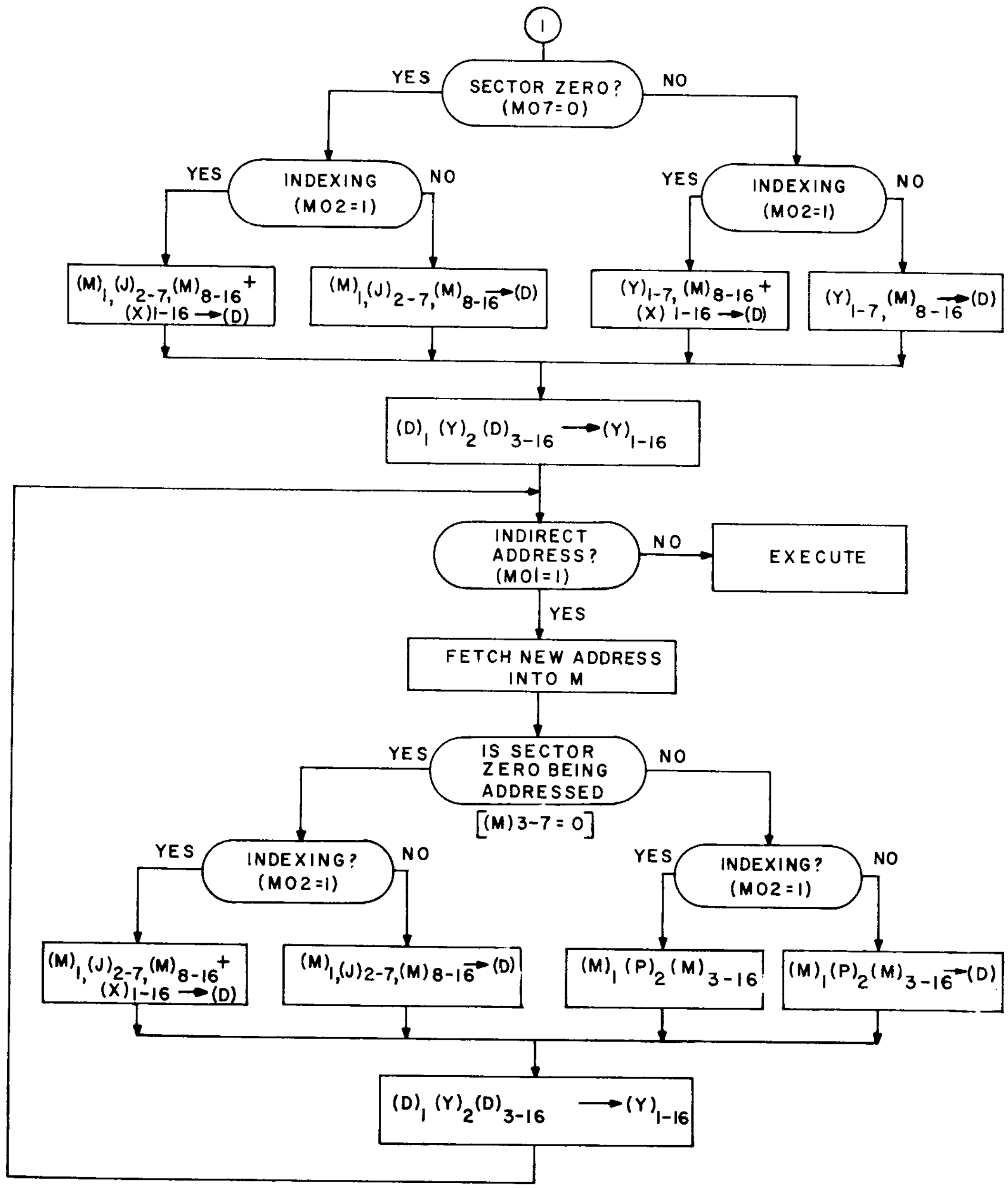


Figure 2-2. Operation of a System with Memory Lockout and up to 32K of Memory (Sheet 2 of 3)

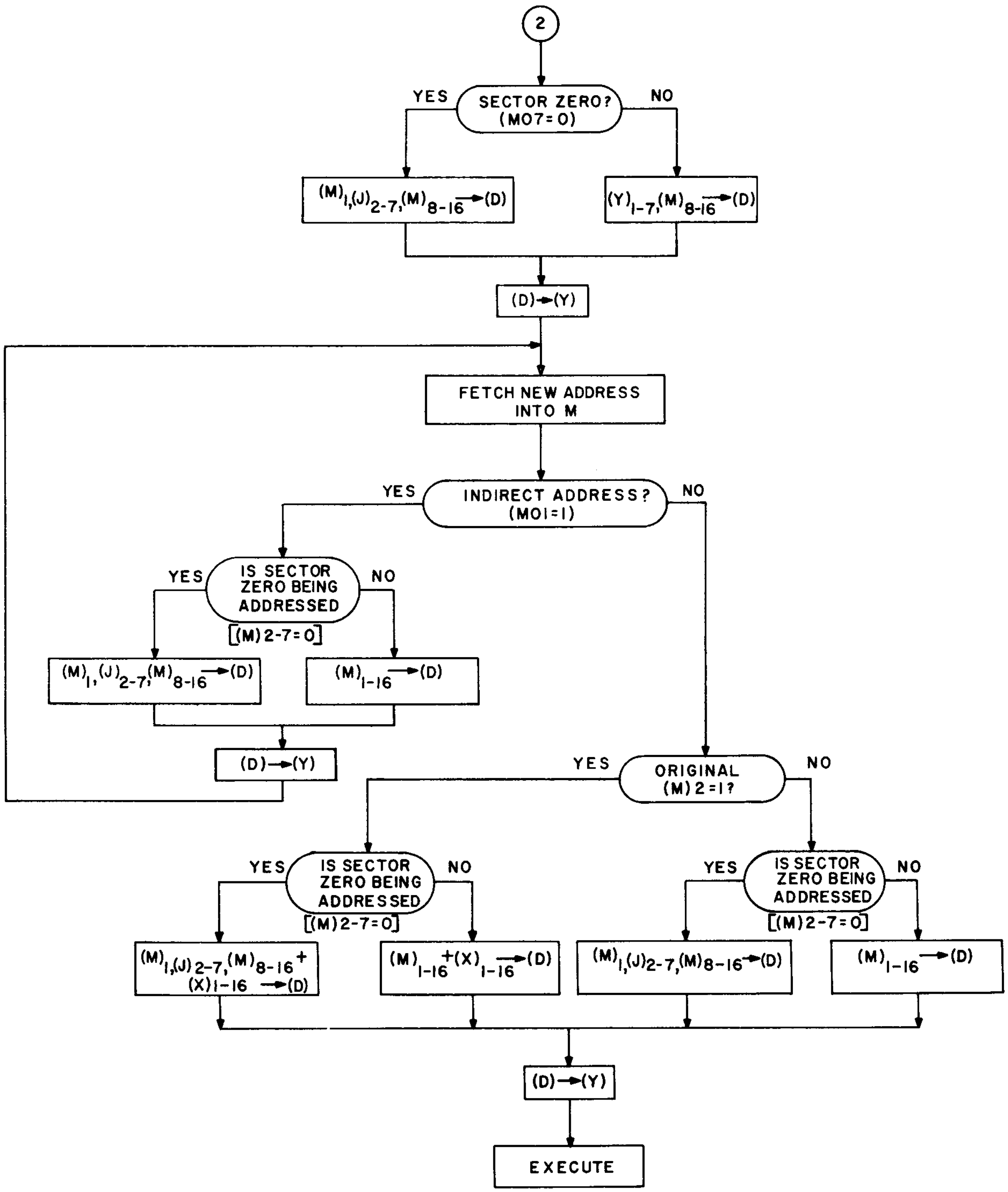


Figure 2-2. Operation of a System with Memory Lockout and up to 32K of Memory (Sheet 3 of 3)

c. If a standard or priority interrupt was pending during the violating instruction, it is processed at the end of the violating instruction and no memory lockout violation interrupt is processed.

An HLT instruction generates a memory lockout violation and is treated as an NOP. The memory lockout violation interrupt is processed in the same manner as STA, STX, etc.

An INH instruction causes a memory lockout violation but does inhibit any standard or priority interrupt pending during its execution. The processing of the memory lockout violation interrupt is the same as STA, STX, etc., except case "c" does not apply.

Table 2-12.
Memory Lockout Instructions

Mnemonic	Type	Instruction Word	Definition	Description	No. of Cycles	Time (μs)
ERM	G	001401	Enter Restricted Mode.	Enables program interrupt and puts computer in restricted mode operation. Restricted operation continues until any program interrupt occurs. Does not take effect until after the next instruction is completed.	1	0.96
SMK '1320	I/O	171320	Set Relocation Register.	$(A)_{2-7} \rightarrow (J)_{2-7}$ Defines physical location of all address references to base sector until another SMK '1320 is executed or MASTER CLEAR is activated, or any interrupt occurs.	2	1.92
SMK '1420	I/O	171420	Set Lockout Mask 1.	$(A)1\text{-}16 \rightarrow (LMR)_{1-16}$	2	1.92
SMK '1520	I/O	171520	Set Lockout Mask 2.	$(A)1\text{-}16 \rightarrow (LMR)_{17-32}$	2	1.92
SMK '1620	I/O	171620	Set Lockout Mask 3.	$(A)1\text{-}16 \rightarrow (LMR)_{33-48}$	2	1.92
SMK '1720	I/O	171720	Set Lockout Mask 4.	$(A)1\text{-}16 \rightarrow (LMR)_{49-64}$	2	1.92

The program interrupt to location $(00062)_8$, generated by an aborted instruction, cannot be masked off.

The restricted mode is entered by executing an ERM instruction. Visual indication of restricted mode operation is given through the use of the ML (memory lockout) indication on the console. Operation in the restricted mode is continuous until any program interrupt occurs. The MASTER CLEAR pushbutton places the machine in the normal mode.

The DMA, DMC, Real-Time Clock, and Memory Increment options are unaffected by the restricted mode since they are treated as agents of normal mode programs. This means that they can write in any memory location, even when they are sharing time with a program executed in the restricted mode.

Normal Mode

Normal mode operation is free of any restrictions and a program can execute any instruction in its repertoire.

Protected Sector Selection

Selection of those memory sectors which are to be protected is controlled by a lock-out mask register (LMR). It is a 16-bit register (expandable to 64 bits) in which each bit is associated with one 512-word memory sector. A bit is zero if the corresponding sector is protected. The register is changed by an SMK instruction and cleared with the MASTER CLEAR pushbutton switch. Table 2-13 shows the specific memory ranges protected by SMK '1420, SMK '1520, SMK '1620, and SMK '1720.

Table 2-13.
Protected Memory Ranges

A-Register Bit	SMK '1420	SMK '1520	SMK '1620	SMK '1720
1	00000-00777	20000-20777	40000-40777	60000-60777
2	01000-01777	21000-21777	41000-41777	61000-61777
3	02000-02777	22000-22777	42000-42777	62000-62777
4	03000-03777	23000-23777	43000-43777	63000-63777
5	04000-04777	24000-24777	44000-44777	64000-64777
6	05000-05777	25000-25777	45000-45777	65000-65777
7	06000-06777	26000-26777	46000-46777	66000-66777
8	07000-07777	27000-27777	47000-47777	67000-67777
9	10000-10777	30000-30777	50000-50777	70000-70777
10	11000-11777	31000-31777	51000-51777	71000-71777
11	12000-12777	32000-32777	52000-52777	72000-72777
12	13000-13777	33000-33777	53000-53777	73000-73777
13	14000-14777	34000-34777	54000-54777	74000-74777
14	15000-15777	35000-35777	55000-55777	75000-75777
15	16000-16777	36000-36777	56000-56777	76000-76777
16	17000-17777	37000-37777	57000-57777	77000-77777

NOTE

Locations 00001-00017 are always protected against all programs, restricted or normal. However, no MLO violation interrupt occurs if an attempt is made to write in these locations unless sector zero is protected and the machine is in the restricted mode.

HIGH-SPEED ARITHMETIC UNIT OPTION (HONEYWELL 316/516-11)

This option enhances the arithmetic capability of the computer by providing hardware implementation of multiply, divide, and normalize functions. The option also provides double-word load, store, add, and subtract functions (double precision mode). All multiply, divide, and normalize functions are performed automatically in a double precision mode; a special double precision instruction must precede the performance of standard arithmetic operations if they are to be carried out in a double precision mode. (See Section I, Figure 1-8.)

Six optional instructions are added to the machine complement whenever the high-speed arithmetic option is included in a system, and four instructions (LDA, STA, ADD, and SUB) have their execution modified. The optional instructions are listed and described in Table 2-14.

Instructions which reference double precision operands must produce even effective addresses (after all indirection and indexing). An odd effective address causes the instruction to be executed as if it had the next lower even effective address in the case of double load, add, or subtract. An odd effective address in a double precision store causes the B-register content to be stored in the specified location without affecting any other location. This requirement is automatically taken care of when programs are written in DAP language.

Instruction Complement

Table 2-14 describes the high-speed arithmetic unit instructions.

REAL-TIME CLOCK OPTION (HONEYWELL 316/516-12)

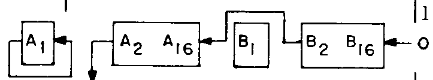
The real-time clock option (RTC) permits the programmer to keep track of real time by automatically incrementing memory location $(00061)_8$. For the DDP-516, the frequency and stability of the incrementation are the same as the primary mainframe power source (50 or 60 ±2 Hz). With a 60-Hz power source the RTC increments location $(00061)_8$ every 16.67 ms; with a 50-Hz source, every 20 ms. Incrementing can be enabled or disabled with an OCP '0020 or OCP '0220 instruction, respectively. For the H316, the incrementing rate is adjustable from 5 to 20 ms, independent of power source.

When memory location $(00061)_8$ overflows from $(177777)_8$ to $(000000)_8$, the RTC causes a program interrupt by means of the standard interrupt line. The program interrupt can be inhibited or enabled with an SMK '0020 instruction. (See Standard Interrupt description.) OTB 16 (A-register bit 16) controls the RTC interrupt. The interrupt can be tested by an SKS '0020 instruction and reset by an OCP '0220 or OCP '0020 instruction. If the RTC tries to interrupt when interrupt is masked off it waits until interrupt is enabled (by a proper SMK '0020 instruction) and then causes an interrupt. Overflow from $(177777)_8$ to $(000000)_8$ does not inhibit incrementing.

Instruction Complement

The addition of the RTC option to a system adds three instructions to the basic system complement. The added instructions are described in Table 2-15.

Table 2-14.
High-Speed Arithmetic Unit Instructions

Mnemonic	Type	Op Code	Definition	Description	No. of Cycles	Time (μs)	
						DDP-516	H316
MPY	MR	16	Multiply	(A) x [EA] → (A, B)	5.5	5.28	8.8
DIV	MR	17	Divide	(A, B) ÷ [EA] → (A) Remainder → (B) Overflow Status → (C) If initial magnitude of dividend is $\geq$ magnitude of divisor overflow occurs	10.0 or 10.5 or 11.0	9.60 or 10.08 or 10.56	16.0 or 16.8 or 17.6
NRM	G	000101	Normalize	 Shift until $(A)_2 \neq (A)_1$; number of shifts required stored as Shift Count	1 + n/2	0.96+ 0.48n	1.6+ 0.8n
SCA	G	000041	Shift Count to A	Shift Count → $(A)_{11-16}$ 0 → $(A)_{1-10}$ The shift count is valid if no IAB, MPY, DIV, OTK, shift, or double precision instruction has been executed since the last NRM instruction was executed.	1	0.96	1.6
DBL*	G	000007	Enter Double Precision Mode	Execute LDA, STA, ADD, and SUB as DLD, DST, DAD and DSB, respectively, until SGL is executed or MASTER CLEAR is depressed	1	0.96	1.6
SGL*	G	000005	Enter Single-Precision Mode	Execute LDA, STA, ADD, and SUB in normal single precision	1	0.96	1.6
DLD	MR	02	Double Precision Load	[EA] → (A) [EA + 1] → (B)	3	2.88	4.8
DST	MR	04	Double Precision Store	(A) → [EA] (B) → [EA + 1]	3	2.88	4.8
DAD	MR	06	Double Precision Add	(A, B) + [EA, EA + 1] → (A, B) Overflow Status → (C) If $[EA + 1]_1 \neq (B)_1$, an invalid sum results	3	2.88	4.8
DSB	MR	07	Double Precision Subtract	(A, B)-[EA, EA + 1] → (A, B) Overflow Status → (C) IF $[EA + 1]_1 \neq (B)_1$, an invalid difference results	3 3	2.88 2.88	4.8 4.8

*See OTK, INK instructions Table 2-2.

Table 2-15.
Real-Time Clock Option Instruction Complement

Mnemonic	Type	Instruction Word	Definition	Description	No. of Cycles	Time (μs)	
						DDP-516	H316
OCP '0220	IO	030220	Reset Program Interrupt Request and Stop Clock	This instruction inhibits the RTC and resets the program interrupt request. One more real time clock break may occur immediately following this instruction if the increment request occurred during the execution of this instruction.	2	1.92	3.2
OCP '0020	IO	030020	Reset Program Interrupt Request and Run Clock	This instruction enables the RTC and resets the program interrupt request. The first increment request occurs within 0 to 16.7 ms following this instruction.	2	1.92	3.2
SKS '0020	IO	070020	Skip if RTC Not Interrupting	If the RTC is not requesting a program interrupt, the computer skips the next instruction.	2	1.92	3.2

DIRECT MULTIPLEX CONTROL (DDP-516)

The direct multiplex control (DMC) option permits data transfer between peripheral devices and the computer memory concurrently with computation.

When a device has data to input, or is ready to accept data, it uses the DMC control lines to request service. Devices request service from the DMC on lines called DIL. DIL line 1 has highest priority, line 16 has lowest. The priority network allows the highest priority line which has its DIL set to be serviced by the next DMC cycle.

When a DMC cycle is required, the DMC sends a break request to the CPU. When the CPU has completed the current instruction, a DMC cycle is executed. During this cycle the appropriate transfer between the device and the memory takes place using the standard I/O bus.

This process is repeated each time the I/O device indicates that it is ready until the required number of words has been transferred. When the required number of words has been transferred, the DMC sends an end-of-range (ERL) signal to the device. The device may use this signal to generate a program interrupt.

Up to 16 channels may be controlled by the DMC. Each channel requires a starting and ending address for the block transfer. These addresses (a pair per channel) are stored in dedicated memory locations which are listed in Table 2-16.

Table 2-16.
DMC Start and Terminal Memory Address Locations (DDP-516)

Channel Number	Starting Address	Ending Address
1	00020	00021
2	00022	00023
3	00024	00025
4	00026	00027
5	00030	00031
6	00032	00033
7	00034	00035
8	00036	00037
9	00040	00041
10	00042	00043
11	00044	00045
12	00046	00047
13	00050	00051
14	00052	00053
15	00054	00055
16	00056	00057

Bit 1 of the starting address is used to specify the input or output mode. A one in bit 1 sets the DMC in the input mode. A zero in bit 1 sets the DMC in the output mode. The remaining 15 bits specify the starting address of the data block. In the input mode, data from the device are stored beginning at this address. In the output mode, data beginning at this address are sent to the device. The high order bit of the final address is not interpreted. The remaining 15 bits specify the address into or out of which the final transfer takes place.

The DMC can effect a transfer following any instruction, provided a DMC request from a device is transmitted to the DMC 0.6 μs before the end of that instruction. If a request occurs less than 0.6 μs before the end of an instruction, the DMC cycle may not occur until after the next instruction.

The data transfer is completed 1.74 μs into the DMC cycle for an input, 3.0 μs for an output. Thus, the longest waiting time, from the time a request occurs to the time the data transfer is completed, is:

$$T_{wc} = T_{1i} + 3.84M + 1.2N + \begin{matrix} 2.34 \text{ (input)} \\ 3.60 \text{ (output)} \end{matrix}$$

where

T_{wc} = worst case waiting time (μs) from request to completion of data transfer.

T_{1i} = execution time of longest* instruction (μs)

*The longest useful instruction in the CPU repertoire is executed in 16.32 μs. (Shifts of more than 32 places and memory reference instructions with more than six levels of indirect addressing are not considered "useful" in this context.) Lower values of T_{1i} may be used to facilitate input/output buffer design, provided appropriate programming constraints are adopted.

M = number of higher priority DMC requests which may occur during T_{wc}.

N = number of DMA requests which may occur during T_{wc}.

Each DMC cycle requires four memory cycles, or 3.84 μs, during which computation is suspended. At 0.6 μs before the end of a DMC cycle, the device request lines are inspected. If a device is requesting at this time, another DMC cycle immediately follows the first. DMC cycles continue as long as requests are waiting. During this time the CPU cannot resume control.

The maximum transfer rate of a single DMC channel is one word every four cycles or 260 kHz. This rate can be attained if this channel is the only channel being used. If the DMC is operating at 260 kHz, no computation can take place. In order to operate between 200 and 260 kHz, T_{1i} must be 0.96 μs, for example an unconditional JMP.

DMC Subchannel

A device is connected to the DMC control unit through a DMC subchannel. The DMC subchannel, available as an option on a number of standard I/O devices, contains the necessary logic to permit the device to operate in the DMC mode.

DMC Auto-Switch Option

The DMC Auto-Switch option provides automatic switching between two DMC subchannels to permit the continuous transfer of data at high speed. To use the Auto-Switch option one DMC subchannel is set up as described above and the data transfer is started. While data are being transferred by the first DMC subchannel, the second DMC subchannel is set up. When the data transfer specified for the first subchannel is complete, the Auto-Switch option automatically switches to the second DMC subchannel and data transfer continues without interruption. An end-of-transmission interrupt occurs on the standard interrupt line to indicate that the switch has been made. The first DMC subchannel must again be set up. When the data transfer specified for the second subchannel is complete, the Auto-Switch option automatically switches back to the first subchannel and interrupts. Switching is accomplished within one DMC cycle. This process is repeated continuously until the device is stopped or taken out of the DMC mode. Indicators associated with the device transferring data may be interrogated by the SKS instruction to determine which channel is active at any time and to determine which channel caused an interrupt.

DIRECT MULTIPLEX CONTROL (H316)

The direct multiplex control (DMC) option permits high speed I/O transfers as requested by peripheral devices. Transfer of data takes place using the standard I/O bus.

The DMC can effect a transfer following any instruction, provided a DMC request transmitted to the DMC on a DIL line has occurred 1.0 μs before the end of that instruction. If a request occurs less than 1.0 μs before the end of an instruction, the DMC cycle may not occur until after the next instruction.

During the execution of a multiple cycle instruction, a request is honored within 6.4 μs provided indirect addressing is limited to one level. Each additional level adds 1.6 μs.

Standard DMC

Two DMC models are available. The standard DMC (H316-20) requires a 4-cycle break for each word transferred.

The data transfer is completed 4.0 μs into the DMC cycle for an input, or 5.05 μs for an output. Thus the longest waiting time from the time a DIL occurs to the time the data transfer is completed is:

$$T_{wc} = T_{li} + 6.4M + \begin{matrix} 4.0 & \text{(Input)} \\ 5.05 & \text{(Output)} \end{matrix}$$

where

T_{wc} = worst-case waiting time (μs) from DIL to completion of data transfer.

T_{li} = execution time of longest* instruction sequence (μs).

*The longest instruction sequence is 6.4 μs provided indirect addressing is limited to one level. Each additional level adds 1.6 μs to the longest instruction sequence.

M = number of higher priority DMC requests which may occur during T_{wc}.

Each standard DMC cycle is four memory cycles, or 6.4 μs during which computation is suspended.

At 1 μs before the end of a standard DMC cycle, the DIL lines are inspected; therefore, if a DIL line is at +6V signifying a DMC request, another DMC cycle immediately follows the first. DMC cycles continue as long as requests are waiting. During this time the CPU cannot resume control.

The maximum transfer rate of a single DMC channel is one word every 4 cycles, provided that only one channel is being used. No computation can take place at this rate.

High-Speed DMC

In the high-speed DMC (H316-21), the first DMC cycle on a channel requires 6.4 μs, while succeeding cycles require 3.2 μs. Computation is suspended during all DMC cycles.

The data transfer for the first DMC cycle on a channel is completed 4.0 μs into the DMC cycle for an input, or 5.05 μs for an output. Data transfers for succeeding cycles are completed 2.0 μs into the DMC cycle for input and 2.65 μs into the DMC cycle for output.

Thus, the longest waiting time, from the time a DIL occurs to the time the data transfer is completed, is:

$$T_{wc} = T_{li} + \begin{matrix} 6.4M \\ 3.2N \end{matrix} + \begin{matrix} 4.0 \text{ first cycle, } 2.0 \text{ succeeding cycles (input)} \\ 5.05 \text{ first cycle, } 2.65 \text{ succeeding cycles (output)} \end{matrix}$$

where

T_{wc} = worst case waiting time (μs) from DIL to completion of data transfer.

T_{li} = execution time of longest* instruction sequence (μs).

*The longest instruction sequence is 6.4 μs provided indirect addressing is limited to one level. Each additional level adds 1.6 μs to the longest instruction sequence.

M = number of higher priority DMC requests requiring first cycle service which may occur during T_{wc}.

N = number of higher priority DMC requests not requiring first cycle service which may occur during T_{wc}.

At 1.0 μs before the end of a DMC cycle, the DIL lines are inspected; therefore, if a DIL line is at +6V signifying a DMC request, another DMC cycle immediately follows the first. DMC cycles continue as long as requests are waiting. Consecutive cycles require two cycles, or 3.2 μs. During this time the CPU cannot resume control.

The maximum transfer rate of a single DMC channel is one word every two cycles or 312 kHz after the first cycle has been completed. This rate can be attained if this channel is the only channel being used. If the DMC is operating at 312 kHz no computation can take place. DMC has priority over the real-time clock, standard interrupt, memory increment, and priority interrupts.

DIL line 1 has highest priority, line 16 the lowest. The priority network allows the highest priority line, which has its DIL at +6V to be serviced by the next DMC cycle.

DMC Subchannel

The DMC control unit is connected to a device through a DMC subchannel. The DMC subchannel, available as an option on many standard I/O devices, contains the necessary logic to permit the device to operate in the DMC mode. It enables the device to control DMC requests and to generate a standard program interrupt when transmission is terminated.

DMC Auto-Switch Option

This option enables a device using DMC to transfer large gapless blocks of data at high speed. Whenever the device is ready for a data transfer, it causes a DIL (N). When End-of-Range is reached, and the device is ready again, a Standard Interrupt occurs and DIL (N+1) also occurs. DIL (N+1) occurs until the next ERL. When ready again, a Standard Interrupt and DIL (N) occur. This cycle repeats continuously until the device is stopped or taken out of DMC mode.

Table 2-17.
DMC Start and Terminal Memory Address Locations (H316)

Channel Number	Starting Address	Ending Address
1	00020	00021
2	00022	00023
3	00024	00025
4	00026	00027
5	00030	00031
6	00032	00033
7	00034	00035
8	00036	00037
9	00040	00041
10	00042	00043
11	00044	00045
12	00046	00047
13	00050	00051
14	00052	00053
15	00054	00055
16	00056	00057

To operate a device, store the starting address (the first bit is 1 for the input mode, and 0 for the output mode) in the assigned location for the starting address. Store the terminal address in its assigned location.

Using appropriate OCPs, set up the device in the input or output mode, and set up the DMC mode; OCP order is defined in each device specification.

If an interrupt is used to detect the end-of-data transmission, the PI mask flip-flop for the device must be set up to a 1, interrupts must be enabled, and the desired interrupt routine must be part of the program.

DIRECT MEMORY ACCESS OPTION (DDP-516)

The direct memory access (DMA) option provides the central processor (CPU) with high speed input/output data transfer paths for addressing up to 32K of memory. The transfer rate is a maximum of one word every 0.96 μs.

The DMA has the highest priority of all system options relative to memory access. The DMA is capable of interrupting between machine cycles such that any DMA interrupt request occurring during any cycle has access to memory at the end of that cycle. The DMA is given access to memory without regard to whether or not the cycle just ended represents the completion of an instruction. These interrupts or breaks are for a minimum of 1200 ns for a single word transfer and 240 ns + N (960 ns) for continuous multi-word word transfers where N is the number of words transferred.

The DMA can effect a transfer following a memory cycle providing the request occurs 0.57 μs before the end of cycle. However, requests arriving any later are serviced after the next memory cycle. The longest time between a request and the completion of the corresponding data transfer is 1.89 μs for input transfers and 2.64 μs for output transfers.

With few exceptions, all computation is momentarily suspended while a DMA cycle is in progress. The exceptions refer to the iterative instruction (e.g., LGL, LLL, LRR, etc.). These instructions comprise the shift/rotate group and the multiply/divide option. The execution of these instructions continues simultaneously with the DMA transfer cycle.

A DMA can have from one to four channels. The channels are arranged in a priority network with channel 1 having the highest priority and channel 4 having the lowest priority.

Each channel has a 16-bit address counter which stores the starting address and a 16-bit range counter which stores the two's complement of the block size. The most significant bit (bit 1) of the starting address is used to specify input or output mode. A one in bit 1 sets the DMA in the input mode. The remaining 15 bits specify the memory address from which the first transfer will occur. The range and address counters are incremented each time a data transfer occurs. Range counter overflow signifies the completion of a block transfer. This is accomplished by the generation of an end-of-range signal which is sent to each device and can be used to cause a program interrupt. The contents of the range counters can be read into the computer to determine whether an external stop signal has terminated the DMA operation before the specified number of words are transferred.

Instruction Complement

A listing of the instructions required for use with the DMA option is presented in Table 2-18.

The programming sequence for operating a device is:

a. Load Address Counter for Specific Channel (this will also clear the range register).

b. Load Range Register with two's complement of number of words to be transferred.

c. Activate Device.

Table 2-18.
Direct Memory Access Instructions

Mnemonic	Type	Instruction Code	Definition	Description	No. of Cycles	Time (μs)
SMK '0124	I/O	170124	Load Address Counter Channel 1	$(A)_{1-16} \rightarrow (AC1)_{1-16}$ $0 \rightarrow (RC1)_{1-16}$	2	1.92
SMK '0224	I/O	170224	Load Address Counter Channel 2	$(A)_{1-16} \rightarrow (AC2)_{1-16}$ $0 \rightarrow (RC2)_{1-16}$	2	1.92
SMK '0324	I/O	170324	Load Address Counter Channel 3	$(A)_{1-16} \rightarrow (AC3)_{1-16}$ $0 \rightarrow (RC3)_{1-16}$	2	1.92
SMK '0424	I/O	170424	Load Address Counter Channel 4	$(A)_{1-16} \rightarrow (AC4)_{1-16}$ $0 \rightarrow (RC4)_{1-16}$	2	1.92
SMK '1124	I/O	171124	Load Range Counter Channel 1	$(A)_{2-16} \vee (RC1)_{2-16}$ $\rightarrow (RC1)_{2-16}$	2	1.92
SMK '1224	I/O	171224	Load Range Counter Channel 2	$(A)_{2-16} \vee (RC2)_{2-16}$ $\rightarrow (RC2)_{2-16}$	2	1.92
SMK '1324	I/O	171324	Load Range Counter Channel 3	$(A)_{2-16} \vee (RC3)_{2-16}$ $\rightarrow (RC3)_{2-16}$	2	1.92
SMK '1424	I/O	171424	Load Range Counter Channel 4	$(A)_{2-16} \vee (RC4)_{2-16}$ $\rightarrow (RC4)_{2-16}$	2	1.92
INA '1124	I/O	13 1124	Read Range Counter Channel 1	If end-of-range, INA = NOP; otherwise, $1 \rightarrow (A)_1$ $(RC1)_{2-16} \rightarrow (A)_{2-16}$ and skip next instruction	2	1.92
INA '1224	I/O	13 1224	Read Range Counter Channel 2	If end-of-range, INA = NOP; otherwise, $1 \rightarrow (A)_1$ $(RC2)_{2-16} \rightarrow (A)_{2-16}$ and skip next instruction	2	1.92
INA '1324	I/O	13 1324	Read Range Counter Channel 3	If end-of-range, INA = NOP; otherwise, $1 \rightarrow (A)_1$ $(RC3)_{2-16} \rightarrow (A)_{2-16}$ and skip next instruction	2	1.92
INA '1424	I/O	13 1424	Read Range Counter Channel 4	If end-of-range, INA = NOP; otherwise, $1 \rightarrow (A)_1$ $(RC4)_{2-16} \rightarrow (A)_{2-16}$ and skip next instruction	2	1.92

DMA Auto-Switch

The DMA Auto-Switch option functions in a manner analogous to that previously described for the DMC Auto-Switch option.

PRIORITY INTERRUPT OPTION (HONEYWELL 316/516)

A multi-level priority interrupt system eliminates the need for an interrupt service routine to determine which one of the available interrupt lines caused an interrupt. A unique memory location is dedicated to each interrupt line. These locations are used in the same manner as the standard interrupt location in the standard interrupt system. When an interrupt occurs, the computer generates an indirect jump and store location instruction (JST) referencing the memory location dedicated to the source of the interrupt. Execution time of the computer-generated JST instruction is three cycles unless bit 1 of the dedicated location is a one. A one in this bit location indicates further indirect addressing; an additional cycle is required for each additional level of indirect addressing. Included in the option is a mask register which permits individual interrupt lines to be enabled and disabled under program control. This permits the relative priority of the interrupt lines to be established by the programmer.

The interrupt option is provided in groups of four interrupt lines. Up to 12 groups or a total of 48 interrupt lines can be handled by the system. The interrupt lines are consecutively numbered and have decreasing priority with increasing number. The standard interrupt line is designated line 0 and retains its standard location $(63)_8$. The dedicated locations for the optional interrupt lines are shown in Table 2-19. On systems with more than 16K of memory the occurrence of a program interrupt causes the CPU to go into the extend mode. (See extended addressing for details of operation in this mode.)

Table 2-19.
Dedicated Locations for the 12 Groups
of Priority Interrupt Lines

Priority Interrupt Group	Dedicated Locations (Octal Codes)
1	00064 - 00067
2	00070 - 00073
3	00074 - 00077
4	00100 - 00103
5	00104 - 00107
6	00110 - 00113
7	00114 - 00117
8	00120 - 00123
9	00124 - 00127
10	00130 - 00133
11	00134 - 00137
12	00140 - 00143

Priority Interrupt Control

Program interrupts requested by Priority Interrupt lines are individually controlled by mask bits associated with each group of interrupt lines. In addition, all Priority Interrupt lines are controlled by the INH and ENB instructions. Priority interrupt is inhibited until an ENB instruction has been executed. Following the execution of an ENB instruction, an interrupt is accepted on any interrupt line having its associated mask bit set (one). Interrupt remains enabled until an INH instruction is executed or an interrupt occurs on any enabled line (forced INH). Following an interrupt or the execution of an INH instruction, interrupts are inhibited until an ENB instruction is executed.

The mask bits associated with each group of interrupt lines are controlled by SMK '0X20 instructions. These instructions set the appropriate bit in the mask register if the corresponding bit in the A-register is a one and reset the mask register bit if the corresponding A-register bit is a zero. Table 2-20 shows the mask assignments for the optional interrupt lines and the SMK instructions that service them.

NOTE

If an interrupt request occurs during the execution of an SMK instruction disabling that interrupt, the interrupt may or may not be accepted (depending on the exact timing of the interrupt signal with respect to the execution of the SMK instruction); therefore, the interrupt mask register should be changed only when interrupt is inhibited.

Table 2-20.
Priority Interrupt Mask Assignments

A-Register Bit No.	SMK '0120	SMK '0220	SMK '0320
1	1	17	33
2	2	18	34
3	3	19	35
4	4	20	36
5	5	21	37
6	6	22	38
7	7	23	39
8	8	24	40
9	9	25	41
10	10	26	42
11	11	27	43
12	12	28	44
13	13	29	45
14	14	30	46
15	15	31	47
16	16	32	48

Interrupt Line Number

MEMORY INCREMENT (HONEYWELL 316/516-26)

Groups of four priority interrupt lines may be optionally changed to memory increment break lines. Any number of priority interrupt groups may be so modified. However, the modified groups must be consecutive starting with the first group of four lines.

The function performed by a memory increment break is:

$$[\text{dedicated location}] + 1 \rightarrow [\text{dedicated location}]$$

There is no overflow indication and no interrupt generated on overflow. Execution of the break requires three cycles.

NOTE

Memory increment requests are not subject to control by the INH or ENB instructions; however, mask register bits are associated with memory increment lines for individual line control as described under Priority Interrupt Control, preceding. This interrupt does not cause the CPU to go into the extend mode.

SECTION III
INPUT/OUTPUT CHANNELS AND DEVICES

The Honeywell 316/516 system includes a variety of I/O devices. The optional devices, as well as the standard, can be used in a number of ways. They can be programmed by using FORTRAN IV, standard I/O library subroutines, or special purpose user-prepared DAP programs. The FORTRAN IV manual (Doc. No. 130071364) contains FORTRAN I/O statements and format specifications. Users who would like to operate I/O devices using standard I/O library subroutines can find complete documentation in the Honeywell 316/516 Operators Guide (Doc. No. 70130072165). Those who wish to prepare their own special purpose I/O programs can find information in the following pages. Included in this section are discussions of:

- ASR-33/35, Model 316/516-53/55, 316/516-56
- High-Speed Paper Tape Reader, Model 316/516-50
- High-Speed Paper Tape Punch, Model 316/516-52
- Card Reader, Model 316/516-61

Discussions of additional devices may be obtained from marketing representatives as required by user options. They include:

- Parallel I/O Channels, Model 316/516-32/33/34
- SKS/OCP, Model 316/516-29
- Line Printer, Model 316/516-7050
- Magnetic Tape System, Model 316/516-4100
- Fixed Head Disc File, Model DDP-516-4400
- Moving Head Disc File, Model 316/516-4600
- Process Interface Controller (PIC), Model DDP-516-8100

ASR-33/35 TELETYPE UNITS (HONEYWELL 316/516-53/55, 316/516-56)

The ASR-33/35 Teletype Unit is the basic I/O device for the Honeywell 316/516 computer. The ASR-33/35 is a versatile device that prints out data from the computer or transmits data to the computer from the keyboard at the rate of 10 characters per second. It can also read and punch paper tape at the same rate. In the local mode the unit may be used for off-line paper tape preparation, reproduction, or listing.

Keyboard and Carriage Features

The ASR-33/35 keyboard is similar to that of a standard typewriter. The keyboard contains four rows of keys that generate an eight-level internal code. (See Figure 3-1 and Table 3-1). Letters and numerals are transmitted without a shift, similar to lower case transmissions on a typewriter. Special characters (?, =, *, etc.) are typed by using the shift key, similar to upper case positions on certain typewriters. Control functions, generated by using the control (CTRL) key, are X-OFF (S key), X-ON (Q key), EOM (C key), and BELL (G key). The LINE FEED and RETURN codes are transmitted without the CTRL key being depressed.

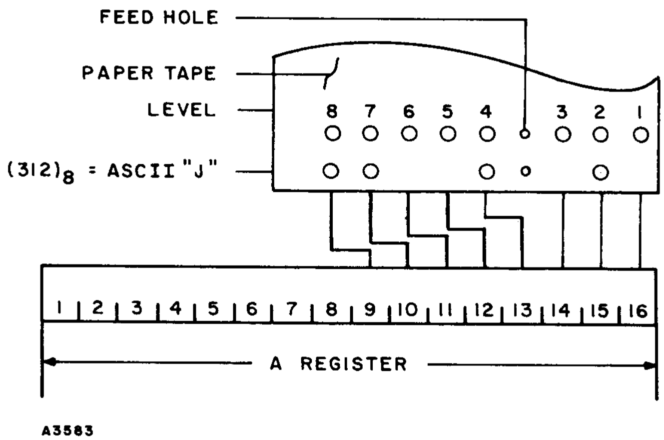


Figure 3-1. ASR-33/35 Paper Tape Format

The ASR-33/35 can print up to 72/75 characters per line. A carriage return and line feed must be executed after the last character to be printed in each line.

The ASR-33/35 keyboard is interlocked for all keys except the SHIFT, CTRL, and REPT keys, preventing more than one key being depressed at one time. The keyboard does not lock in the upper case position so the operator must hold the SHIFT key depressed to produce special (upper case) characters.

Table 3-1.
ASR-33/35 Character and Symbol Codes

Code	Page Printer	Code	Page Printer	Code	Page Printer	Code	Page Printer
000	Null *	040	Space	100	@	140	@
001	Null	041	!	101	A	141	A
002	Null	042	"	102	B	142	B
003	Null	043	#	103	C	143	C
004	Null	044	$	104	D	144	D
005	WRU	045	%	105	E	145	E
006	Null	046	&	106	F	146	F
007	Bell	047	'	107	G	147	G
010	Null	050	(	110	H	150	H
011	Null	051	)	111	I	151	I
012	LF	052	*	112	J	152	J
013	Null	053	+	113	K	153	K
014	Null	054	,	114	L	154	L
015	Null	055	-	115	M	155	M
016	Null	056	.	116	N	156	N
017	Null	057	/	117	O	157	O
020	Null	060	0	120	P	160	P
021	Null	061	1	121	Q	161	Q
022	Null	062	2	122	R	162	R
023	Null	063	3	123	S	163	S
024	Null	064	4	124	T	164	T
025	Null	065	5	125	U	165	U
026	Null	066	6	126	V	166	V
027	Null	067	7	127	W	167	W
030	Null	070	8	130	X	170	X
031	Null	071	9	131	Y	171	Y
032	Null	072	:	132	Z	172	Z
033	Null	073	;	133	[	173	[
034	Null	074	<	134	\	174	Null
035	Null	075	=	135	]	175	Null
036	Null	076	>	136	↑	176	Null
037	Null	077	?	137	←	177	Null

* Whenever the HERE-IS key is depressed (available on ASR-33 only), the answer back drum is activated, producing a burst of 20 characters of all zeros.

Table 3-1. (Cont)
ASR-33/35 Character and Symbol Codes

Code	Page Printer	Key Depressed		
		Lower Case	Simult. Control	Simult. Shift and Control
200	Null			P
201	Null		A	
202	Null		B	
203	Null		C	
204	Null		D	
205	WRU		E	
206	Null		F	
207	Bell		G	
210	Null		H	
211	Null		I	
212	LF	LF	J	
213	Null		K	
214	Null		L	
215	CR	CR	M	
216	Null		N	
217	Null		O	
220	Null		P	
221	X-ON		Q	
222	TAPE		R	
223	X-OFF		S	
224	Null		T	
225	Null		U	
226	Null		V	
227	Null		W	
230	Null		X	
231	Null		Y	
232	Null		Z	
233	Null			K
234	Null			L
235	Null			M
236	Null			N
237	Null			O

Table 3-1. (Cont)
ASR-33/35 Character and Symbol Codes

Code	Page Printer	Key Depressed			
		Lower Case	Simult. Shift	Simult. Control	Simult. Shift and Control
240	Space	Space Bar		Space Bar	
241	!		1		1
242	"		2		2
243	#		3		3
244	$		4		4
245	%		5		5
246	&		6		6
247	'		7		7
250	(		8		8
251	)		9		9
252	*		:		:
253	+		;		;
254	,	,		,	
255	-	-(minus)		-(minus)	
256	.	.		.	
257	/	/		/	
260	0	0		0	
261	1	1		1	
262	2	2		2	
263	3	3		3	
264	4	4		4	
265	5	5		5	
266	6	6		6	
267	7	7		7	
270	8	8		8	
271	9	9		9	
272	:	:		:	
273	;	;			
274	<		,		,
275	=		-	Alt Mode	-
276	>		.		.
277	?		/	Rub Out	?

Table 3-1. (Cont)
ASR-33/35 Character and Symbol Codes

Code	Page Printer	Key Depressed Lower Case	Key Depressed Simult. Shift
300	@		P
301	A	A	
302	B	B	
303	C	C	
304	D	D	
305	E	E	
306	F	F	
307	G	G	
310	H	H	
311	I	I	
312	J	J	
313	K	K	
314	L	L	
315	M	M	
316	N	N	
317	O	O	
320	P	P	
321	Q	Q	
322	R	R	
323	S	S	
324	T	T	
325	U	U	
326	V	V	
327	W	W	
330	X	X	
331	Y	Y	
332	Z	Z	
333	[		K
334	\		L
335	]		M
336	↑		N
337	←		O

Table 3-1. (Cont)
ASR-33/35 Character and Symbol Codes

Code	Page Printer	Code	Page Printer	Lower Case
340	@	360	P	
341	A	361	Q	
342	B	362	R	
343	C	363	S	
344	D	364	T	
345	E	365	U	
346	F	366	V	
347	G	367	W	
350	H	370	X	
351	I	371	Y	
352	J	372	Z	
353	K	373	[	
354	L	374	Null	
355	M	375	Null	
356	N	376	Null	
357	O	377	Null	Rub Out

NOTES:

1. Whenever the BREAK key is depressed, a 000 code is generated as long as the key is held depressed. However, when the key is released, an indeterminate character is produced.
2. The symbols appearing in the Page Printer column indicate the reaction of the printer to codes received on the line in output mode and to codes generated by the reader or keyboard in the input mode. Null indicates no printing and no spacing.
3. No entry in the Key Depressed column indicates inability of the keyboard to produce that code.
4. The punch perforates all codes transmitted in input or output mode. On the ASR-35 only, the first Tape On character and its associated rub-out character, if character buffering is used for automatic punch control, are not punched.

ASR-33/35 On-Line Operating Modes

There are two basic modes of operation for the ASR-33/35 when on-line: input mode and output mode. These are set up by the appropriate OCP instruction. Once set up, the ASR-33/35 remains in a given mode until it is changed by another OCP or master clear.

Input Mode. -- The input mode is used to transmit information from the ASR-33/35 keyboard to the computer or from the reader to the computer. In either case, printed copy is produced if the 8-bit character is printable, and a control function is performed if the 8-bit character is a control character. (See Appendix B.) If characters are being read from the reader, any of the 256 possible 8-bit characters appearing on the tape will be transmitted to the computer. When an X-OFF is read, the reader stops after reading the character (two characters for ASR-35 except as described later) following the X-OFF, unless that following character is an X-ON. MASTER CLEAR places the ASR-33/35 in the input mode.

Output Mode. -- The output mode is used to transmit information from the computer to the ASR-33/35 printer or the printer and the punch. In either case, printed copy is produced if the 8-bit character is printable, and a control function is performed if the 8-bit character is a control character. When punching, any 8-bit code transmitted from the computer is punched whether it is printable or not. However, certain 8-bit codes -- $(221)_8$, $(021)_8$, $(005)_8$, and $(205)_8$ -- when transmitted from the computer also cause a control action by the ASR-33/35 and prevent proper transmission of further characters. X-ON, $(221)_8$ or $(021)_8$, starts the paper tape reader and WRU, $(205)_8$ or $(005)_8$, triggers the answer-back drum.

Character Modes

In either the input or output mode, either of two character modes, ASCII or binary, may be used. Code type is selected by individual INA or OTA instructions and may be intermixed in any manner (though this is not normally done).

ASCII Mode. -- In the ASCII mode, a full 8-bit character is transmitted between the least significant 8 bits of the A-register and the ASR-33/35. This permits transmission of any standard character or control character from the reader or keyboard of the ASR-33/35 to the computer or from the computer to the printer or punch.

Binary Mode. -- In the binary mode, a 6-bit character is transmitted to or from the least significant 6 bits of the A-register and the ASR-33/35. In the case of output in the binary mode, an additional 2 bits are automatically added in the high-order position to the 6-bit character to form a printable 8-bit character rather than a control character. On input, the two high-order bits of the 8-bit character transmitted by the ASR-33/35 are stripped and ignored.

ASR-33 Operation

Reader Control. -- The reader can be started under program control as follows:

a. Enable output mode with OCP '0104.
b. Output an X-ON character (221_8) using OTA '0004.
c. Delay until not busy (test with SKS '0104).
d. Enable in input mode with OCP '0004.

Manual starting is controlled with the START/STOP switch. The first character to be read when the reader has been started is the one initially positioned over the read pins.

When operating under manual or program control, the reader stops upon recognition of an X-OFF character. The X-OFF and one following character are transmitted to the device buffer before the reader stops. Manual stopping is controlled by the START/STOP switch. The reader stops automatically if it runs out of paper tape. An X-OFF character stops the reader when reading tape off-line. To continue reading, the operator can restart the reader by depressing the START switch.

Punch Control. -- The punch is controlled by manual operation of the punch ON/OFF switch. When the punch is on, any input from or output to the ASR-33 causes the tape to be punched. Tape leader can be generated in bursts of 20 sprockets with each depression of the HERE-IS key.

Off-Line Operation. -- Off-line operation of the ASR-33 includes the following.

a. Keyboard to printer
b. Keyboard to printer and punch
c. Reader to printer
d. Reader to printer and punch

ASR-35 Operation

The ASR-35 operates in either on- or off-line modes as described in the following paragraphs.

Off-Line. --

K Mode	Keyboard to printer
KT Mode	Keyboard to printer and punch Reader to printer and punch
T Mode	Keyboard to punch Reader to printer

On-Line. --

K Mode	Input transfer from keyboard monitored by printer (ASCII) Output transfer to printer (ASCII)
KT Mode	Input transfer from keyboard monitored by printer and punch if enabled (ASCII) Input transfer from reader monitored by printer and punch if enabled (ASCII or binary) Output transfer to printer and punch if enabled (ASCII or binary)
T Mode	Input transfer from reader monitored by printer (ASCII or binary) Output transfer to printer (ASCII) Simultaneous off-line operation of keyboard to punch (ASCII)
TTS Mode	Input transfer from reader (any eight-level code). Manual control of reader only. Automatic start and stop code inoperative. Simultaneous off-line operation of keyboard to punch (ASCII).
TTR Mode	Output transfer to punch (any eight-level code).

Reader Control (KT and T Modes Only). -- To start the reader under program control, the program must output an X-ON character (021_8 or 221_8). After waiting until the ASR is not busy, an OCP '0004 should be issued to enable the ASR in the input mode before proceeding with input transfer instructions. The reader can also be started by depressing the START switch and rotating the reader manual control switch to the ON position. When started, the first character to be read is the one positioned over the read pins.

To manually start the reader and read under program control (procedure for loading self-loading programs), position the ASR mode switch to the K position, depress the CTRL key, and while CTRL is depressed, depress the Q key. This gives an X-ON to the reader. Position the Mode Control Switch to KT or T mode (T mode suppresses punching). Start the program and move the reader control switch to the RUN position.

When operating under program control, the reader stops two characters after the recognition of an X-OFF character (023_8 or 223_8). The X-OFF character is read into the character buffer, and the next two characters are also read into the character buffer before the tape stops (unless the character following the X-OFF is RUBOUT, in which case only the RUBOUT is read). When operating under program control, the reader cannot be stopped manually or by an X-OFF when operating off-line.

Punch Control (KT Mode, On-Line Only). -- To enable the punch when operating under program control, the program must output a TAPE character (022_8 or 222_8). If reader reads TAPE (022_8 or 222_8), the punch is enabled without program output. A RUBOUT character or a time delay equal to one character time must follow the TAPE character. (The RUBOUT character is not punched on paper tape.) Additional output transfers are then punched on paper tape as required.

To stop the punch when operating under program control, the program must output an X-OFF character followed by a RUBOUT character. Both characters are punched on tape.

Tape leader can be generated off-line on the ASR-35 by depressing the BREAK button until the required amount of leader is punched. The operator may then depress the BACKSPACE and RUBOUT keys to produce a frame of all ones in place of the indeterminate frame produced when the BREAK button is released.

Programming

The control codes assigned to the ASR-33/35 are described in the following paragraphs. In summary they are:

OCP '0004	Enable ASR-33/35 in input mode
OCP '0104	Enable ASR-33/35 in output mode
SKS '0004	Skip if ASR-33/35 is ready
SKS '0104	Skip if ASR-33/35 is not busy
SKS '0404	Skip if ASR-33/35 is not interrupting
SKS '0504	Skip if stop code was not read on ASR-33/35
INA '0004	Input in ASCII mode if ready
INA '0204	Input in binary mode if ready
INA '1004	Clear A and input in ASCII mode if ready
INA '1024	Clear A and input in binary mode if ready
OTA '0004	Output in ASCII mode if ready
OTA '0204	Output in binary mode if ready
SMK '0020	Set interrupt mask

Enable ASR-33/35 in Input Mode (OCP '0004). -- This instruction sets up the device interface to accept characters from the ASR-33/35. It should be given any time it is desired to switch the ASR-33/35 from the output to the input mode. This instruction must not be given while the ASR-33/35 is busy. An SKS '0104 test should precede this instruction.

Enable ASR-33/35 in Output Mode (OCP '0104). -- This instruction sets up the device interface to transmit characters to the ASR-33/35. The instruction must be given any time it is desired to switch from the input to the output mode. The instruction must not be given while the ASR-33/35 is busy. An SKS '0104 test should precede this instruction.

Skip if ASR-33/35 Is Ready (SKS '0004). -- This instruction tests whether the ASR-33/35 device interface is ready to accept another character from the computer or to present another character to the computer.

Skip if ASR-33/35 Is Not Busy (SKS '0104). -- The ASR-33/35 busy signal is defined as follows:

a. In the output mode the ASR-33/35 is busy from the time a character is transmitted from the computer to the ASR-33/35 device interface until it has been serially shifted out to the ASR-33/35. This time is approximately 105 ms.

b. In the input mode the ASR-33/35 is busy from the time the ASR-33/35 starts to serially transfer a character to the device interface until the transfer is complete and the ASR-33/35 ready condition is present. This time is approximately 100 ms.

Skip if ASR-33/35 Is Not Interrupting (SKS '0404). -- This instruction tests whether the ASR-33/35 has caused an interrupt on the standard interrupt line.

Skip if Stop Code Was Not Read on ASR-33/35 (SKS '0504). -- This instruction tests whether a stop code (223_8 or 023_8) has been read by the ASR-33/35. The stop code indication can be tested as soon as the stop code has been read from the ASR-33/35 into the device buffer and is ready for input to the computer. When a stop code is read by an ASR-33/35, the stop code and one/two following characters are transferred to the device buffer before the reader stops. The stop code indication remains present until the character following the stop code is ready for input to the computer (approximately 100 ms).

Input in ASCII Mode If Ready (INA '0004). -- This instruction transmits the full 8-bit character from the ASR-33/35 to the eight least significant bits of the A-register. The A-register is not cleared. Ready must be honored within 1 ms to ensure transmission. If Ready is true, the instruction is executed and the next instruction skipped. If Ready is not true, this instruction is treated as an NOP.

Input in Binary Mode If Ready (INA '0204). -- This instruction transmits the six least significant bits of the 8-bit ASR-33/35 character to the six least significant bits of the A-register. The A-register is not cleared. Ready must be honored within 1 ms to ensure transmission. If Ready is true, this instruction is executed and the next instruction skipped. If Ready is not true, this instruction is treated as an NOP.

Clear A and Input in Binary Mode If Ready (INA '1004). -- Same as INA '0004 except A is cleared before character is transmitted.

Clear A and Input in Binary Mode If Ready (INA '1204). -- Same as INA '0204 except A is cleared before character is transmitted.

Output in ASCII Mode If Ready (OTA '0004). -- This instruction transmits the eight least significant bits of the A-register to the ASR-33/35. If the ASR-33/35 is punching, it punches all eight bits of the code that is transmitted. However, in printing, it determines the character to be printed or the control function to be performed from the seven least significant bits.

Output in Binary Mode If Ready (OTA '0204). -- This instruction transmits the eight least significant bits of the A-register to the ASR-33/35 and then modifies channel 7 (normally A10) to be the inverse of A11. Thus, if the eight least significant bits in the A-register were $(XX1XXXXX)_2$, they would be transmitted to the ASR-33/35 as $(X01XXXXX)_2$. If they were $(XX0XXXXX)_2$, they would be transmitted as $(X10XXXXX)_2$.

Set Interrupt Mask (SMK '0020). -- The A-register bit assignment for the ASR-33/35 is bit 11. This instruction sets the standard interrupt mask flip-flop if the A-register bit is one and resets the mask flip-flop if the bit is a zero.

Sample Program

The following subroutine is intended as an example only. When it is called, the subroutine outputs one character to the ASR-33. The character is printed if it is printable. If the punch is on, the character is punched whether it is printable or not. The subroutine is entered with the character to be outputted in the A-register.

```
        SUBR   ASRTYP, STRT   Subroutine name
        REL
STRT    DAC    **             Subroutine entry point
        SKS    '104           Test ASR busy
        JMP    *-1            Delay until not busy
        OCP    '104           Enable output mode
        OTA    4              Output character in ASCII mode
        JMP    *-1            Delay if ASR not ready
        JMP    STRT           Return to calling program
        END
```

HIGH-SPEED PAPER TAPE READER OPTION (HONEYWELL 316/516-50)

A high-speed, unidirectional perforated tape reader consists of a paper tape reader and the control logic that is required for operational compatibility. The reader employs a pinch roller capstan and brake solenoid system to control tape movement. The control logic includes an eight-bit buffer register that enables transfers via the I/O bus of one frame per computer word. The reader reads eight data channels per frame at the rate of 30 inches per second. With a density of 10 frames per inch, the rate is 300 frames per second.

Loading Procedure

The reader uses standard paper or mylar tapes (black paper recommended) 0.004 to 0.005 in. thick. The tape can be loaded without removing power by rotating a front-mounted READY-LOAD switch clockwise to the LOAD position. The tape must be placed with the three-channel side flush with the inboard guide. After the tape has been loaded, the READY-LOAD switch must be rotated counterclockwise to the READY position.

Programming

The reader operates continuously when reading is initiated with an OCP '0001. Data is transferred to the buffer until the complete tape has been read or until an OCP '0101 is executed.

The control codes assigned to the high-speed paper tape reader are described in the following paragraphs. In summary they are as follows.

OCP '0001	Start reader
OCP '0101	Stop reader
SKS '0001	Skip if tape reader ready

SKS '0401	Skip if tape reader not interrupting
INA '0001	Input from paper tape if ready
INA '1001	Clear A and input from paper tape if ready
SMK '0020	Set interrupt mask

Start Reader (OCP '0001). -- This instruction starts tape motion. The first character to pass the read station is transferred to the device buffer for transmission to the central processor. An interval of 5 ms is required to reach full operating speed after execution of OCP '0001.

Stop Reader (OCP '0101). -- This instruction stops tape motion. This instruction must be executed within 1 ms after a character-ready signal to avoid losing the character after a restart.

Skip if Tape Reader Ready (SKS '0001). -- This instruction skips if the tape reader is in a ready status. The tape reader is ready when a character is available in the device buffer.

Skip if Tape Reader Not Interrupting (SKS '0401). -- The tape reader is interrupting when a character is available and the interrupt mask flip-flop is set.

Input From Paper Tape Reader if Ready (INA '0001). -- Execution of this instruction causes a frame to be ORed into the eight least significant bit positions of the A-register with channel 1 of the frame corresponding to bit position 16. The next program instruction is skipped upon execution of this instruction. If the Ready is not true, this instruction is treated as an NOP.

Clear A and Input From Paper Tape Reader if Ready (INA '1001). -- This instruction is identical to INA '0001 except that the A-register is cleared before the character is transferred in.

Set Interrupt Mask (SMK '0020). -- The A-register bit assignment for the paper tape reader is bit 9. This instruction sets the standard interrupt mask flip-flop if the A-register bit is a one and resets the mask flip-flop if the bit is a zero.

Sample Program

The following subroutine is intended as an example only. When called, it reads two frames from the high-speed paper tape reader and packs the data read into one word. The packed word is left in the A-register upon return to the calling program.

```
         SUBR    PTR      Subroutine name
         REL
PTR      DAC     **       Subroutine entry point
         OCP     1        Start tape reader
         INA     '1001    Clear A and input first frame
         JMP     *-1      Delay until ready
         LGL     8        Shift to pack
         INA     1        Input second frame
         JMP     *-1      Delay until ready
         OCP     '101     Stop reader
         JMP*    PTR      Return
         END
```

NOTE

In the above example, the tape reader was stopped in sufficient time to prevent loss of the following character.

HIGH-SPEED PAPER TAPE PUNCH OPTION (HONEYWELL 316/516-52)

The high-speed paper tape punch consists of a punch unit and the control logic required for interface with the computer. The punch is a synchronous device; pulses generated by a magnetic pickup coil synchronize the interface control circuits. The control logic includes an eight-bit buffer register that receives data transferred from the central processor. The device punches 1-inch, eight-channel paper tape at the rate of 110 frames per second. (Oil impregnated tape is recommended.)

Loading Procedure

a. Thread tape off bottom rear of roll, through wire and roller guides, then to tape guide and punch block.

b. Lead tape between hold-down bar and feed wheel, then out under tape cutter.

c. Apply punch power.

d. Depress feedout lever (located at top center of punch cover), pull the tape to the left until it begins to feed, and then release the feedout lever.

Programming

Punch power can be turned on by means of a switch on the device cabinet or under program control. A 2.5-second interval is required for the device to reach full operating speed after power has been applied. It is suggested that power only be applied under program control except during maintenance or replacement of tape supply.

The control codes assigned to the paper tape reader are described in the following paragraphs. In summary they are:

OCP '0002	Enable paper tape punch
OCP '0102	Turn punch power off
SKS '0002	Skip if punch is ready
SKS '0102	Skip if punch power is on
SKS '0402	Skip if punch is not interrupting
OTA '0002	Output to punch if ready
SMK '0020	Set interrupt mask

Enable Paper Tape Punch (OCP '0002). -- This instruction applies power to the paper tape punch. There is a 5-second delay until the punch is ready to receive data.

Turn Punch Power Off (OCP '0102). -- This instruction removes power from the paper tape punch. Ready status should be tested to be sure that it is ready before execution of this instruction to avoid turning the punch off while data is being punched.

Skip if Punch Is Ready (SKS '0002). -- This instruction skips if ready status is true. Ready status is true when the device buffer is ready to accept new data from the central processor.

Skip if Punch Power Is On (SKS '0102). -- This instruction must precede an OCP '0002. A character might be lost if an OCP '0002 is executed when power is already on.

Skip if Punch Is Not Interrupting (SKS '0402). -- The punch is interrupting when the interrupt mask flip-flop is set and the buffer is ready to receive a character.

Output to Punch if Ready (OTA '0002). -- Execution of this instruction results in an output transfer. If ready status is true, the eight least significant bits of the A-register are transferred to the device buffer and the next instruction is skipped. Ready status then becomes false for approximately 9 ms during which time the contents of the device buffer is punched as a frame, with channel 1 of the frame corresponding to bit position 16 of the A-register.

Set Interrupt Mask (SMK '0020). -- The A-register bit assignment for the paper tape punch is bit 10. This instruction sets the standard interrupt mask flip-flop if the A-register bit is one and resets the mask flip-flop if the bit is zero.

Sample Program

The following subroutine is intended as an example only. When the subroutine is called, it performs one of three functions depending on the entry used. The PON entry is used to apply power to the paper tape punch. It is a separate entry so that the calling program can perform other operations during the 2.5-second interval required for the punch to reach full operating speed. The PNCH entry is used to punch an 80-character card image from a block of 40 words packed two characters per word. The POFF entry is used after the last data block has been punched.

```
        SUBR    PON     Subroutine name for power-on
        SUBR    PNCH    Subroutine name for punch-data
        SUBR    POFF    Subroutine name for power-off
        REL
PON     DAC     **      Power-on entry
        SKS     '102    Test power already on
        OCP     2       If not, turn on
        JMP*    PON     Return
PNCH    DAC     **      Punch-data entry
        LDA     =-40    Set CTR for 80 characters XFER
        STA     CTR     Store CTR
        LDA     BUFA    Get first location of block storage
        STA     LINK    Store in link
LOOP    LDA*    LINK    Get packed data word
        ICA             Set up left character
        OTA     2       Output character
        JMP     *-1     Delay if not ready
        ICA             Set up right character
        OTA     2       Output character
        JMP     *-1     Delay if not ready
        IRS     LINK    Increment storage address
        IRS     CTR     Increment CTR
        JMP     LOOP    Loop to punch next two characters
        JMP*    PNCH    Return
POFF    DAC     **      Power-off entry
        SKS     2       Test ready status
        JMP     *-1     Delay if not ready
        OCP     '102    Turn power off
        JMP*    POFF    Return
BUFA    DAC     COM     Address of packed word storage
LINK    BSS     1       Storage location counter
CTR     BSS     1       Word counter
COM     BSS     40      Packed word storage
        END
```

CARD READER OPTION (HONEYWELL 316/516-61)

The Card Reader includes a 200-cpm card reader and a card reader control unit (CRCU) that contain appropriate interface logic to ensure 16-bit operational compatibility. The card reader is a photoelectric device that reads Hollerith or binary coded punched cards, one at a time, column-by-column, and transmits the data read to the 16-bit interface.

In the Hollerith mode the card reader reads each column as a Hollerith character and converts it to a six-bit code. (See Table 3-2.) The converted characters are then transmitted to the 16-bit interface as bits 11 through 16 of the data word.

In the binary mode the card reader reads each column as a 12-bit byte. Data is transferred to the 16-bit interface in bit positions 5 through 16 of the data word, with bit position 5 represented by row 12 and bit position 16 represented by row 9. This action continues for each column up to and including column 80.

Programming

The control codes assigned to the card reader are:

OCP '0005	Read one Hollerith card.
OCP '0105	Read one binary card.
SKS '0005	Skip if card reader ready.
SKS '0105	Skip if card reader not busy.
SKS '0205	Skip if not end of file.
SKS '0305	Skip if card reader operational.
SKS '0405	Skip if card reader not interrupting.
INA '0005	Input from card reader if ready.
INA '1005	Clear A-register and input from card reader if ready.
SMK '0020	Set interrupt mask.

Read One Hollerith Card (OCP '0005). -- This OCP causes the card reader to feed one card and enables 6-bit Hollerith encoded characters to be read into the A-register with an INA 'X005 instruction.

Read One Binary Card (OCP '0105). -- This OCP causes the card reader to feed one card and allows 12-bit binary data to be read into the A-register with an INA 'X005 instruction.

NOTE

Proper operation of the system is not guaranteed if these OCP's are issued when the card reader is busy.

Skip if Card Reader Ready (SKS '0005). -- This SKS skips if the control unit is ready to send a character to the input bus. The first Ready occurs approximately 110 ms after an OCP read instruction is received. Subsequent Readys occur every 2.4 ms.

Skip if Card Reader Not Busy (SKS '0105). -- This SKS skips if the card reader is not busy. The card reader is busy from the time OCP '0005 or OCP '0105 is received until 1 ms after the 80th column of the card has been read (a total duration of approximately 300 ms).

Table 3-2.
Card Codes

Octal ⚠1	Card (Hollerith)	Character	Octal	Card (Hollerith)	Character
00	All Other Codes		40	11	-
01	1	1	41	11-1	J
02	2	2	42	1-2	K
03	3	3	43	11-3	L
04	4	4	44	11-4	M
05	5	5	45	11-5	N
06	6	6	46	11-6	O
07	7	7	47	11-7	P
10	8	8	50	11-8	Q
11	9	9	51	11-9	R
12	8-2	Null	52	11-0	;
00	0	0			
13	8-3	=	53	11-8-3	$
14	8-4	1	54	11-8-4	*
15	8-5	:	55	11-8-5	[
16	8-6	!	56	11-8-6	End-of-File
17	8-7	>	57	11-8-7	<
20	Blank	Space	60	12	+
21	0-1	/	61	12-1	A
22	0-2	S	62	12-2	B
23	0-3	T	63	12-3	C
24	0-4	U	64	12-4	D
25	0-5	V	65	12-5	E
26	0-6	W	66	12-6	F
27	0-7	X	67	12-7	G
30	0-8	Y	70	12-8	H
31	0-9	Z	71	12-9	I
			72	12-0	↑
32	0-8-2	Null			
33	0-8-3	,	73	12-8-3	.
34	0-8-4	(	74	12-8-4	)
35	0-8-5	Null	75	12-8-5	%
36	0-8-6	]	76	12-8-6	\
37	0-8-7	"	77	12-8-7	←

⚠1 **Octal column is 6-bit code generated by card reader.**

Skip if Not End-of-File (SKS '0205). -- This SKS skips if the end-of-file flip-flop is not set. The EOF flip-flop is set by an 11-8-6 punch read in Hollerith mode or by pushing the END OF FILE button on the reader console when the input hopper is empty. It is reset every time a read card OCP is issued or on MASTER CLEAR.

Skip if Card Reader Operational (SKS '0305). -- This SKS skips if the card reader is in an operational state (that is, power on, feed hopper not empty, no card jam, stacker not full, no read-feed or validity errors, and start button depressed). The level indicating that the card reader is not operational cannot be reset by MASTER CLEAR. If it was set by READ CHECK, FEED CHECK or VALIDITY CHECK, which is indicated by the fact that the particular light on the reader console will be on, both the RESET and START buttons have to be pushed in order to reset the level. If it was set by any other condition (input hopper empty or output stacker full), correcting the condition and depressing the START button resets the level. In all the above cases, if the condition which caused the fault in the first place is still present, the level cannot be reset. If a read OCP is issued when the card reader is not operational, the CRCU becomes busy but no card is clutched. If the non-operational status was caused by an empty input hopper, some programs may be resumed by placing cards in the hopper and depressing the START button.

Skip if Card Reader Not Interrupting (SKS '0405). -- This SKS skips if the card reader has not caused an interrupt. This SKS is used when operating with standard interrupt to determine which device is ready to send or receive new data.

Input from Card Reader if Ready (INA '0005). -- If the control unit is not ready to transfer data, the INA is treated as an NOP and the program continues in sequence. When the control unit ready flip-flop has been set, data is transferred to the A-register in the following manner and the next instruction is skipped.

Hollerith Mode. -- The card reader 6-bit output is ORed into the A-register bit positions 11 through 16. Bits 1 through 10 are unchanged. Card reader codes are listed in Table 3-2.

Binary Mode. -- The 12 bits from one card column, rows 12 through 9, are ORed into A-register bit positions 5 through 16. Bits 1 through 4 are unchanged. The device must be serviced within 2.4 ms after a ready signal is received to ensure that each column of data is successfully transferred.

Clear A-Register and Input from Card Reader if Ready (INA '1005). -- This instruction is identical to INA '0005 but the A-register is cleared before data is transferred.

Set Interrupt Mask (SMK '0020). -- The A-register bit assignment for the card reader is bit 12. This instruction sets the standard interrupt mask flip-flop if the A-register bit is a one and resets the mask flip-flop if the bit is a zero.

Operator Controls and Indicators

Pushbutton combination switches and lamps to indicate the status of the card reader are located on the card reader control panel. Front panel controls and indicators are:

Controls	Function
POWER ON	Applies ac power to reader controls and logic.
POWER OFF	Turns off ac power to reader.
START	Sends a ready level to card reader interface, signaling that a read cycle may commence.
STOP	Stops card reader operation.
RESET	Resets all error condition signals except the NOT READY indicator (this signal can only be reset when the start pushbutton is depressed).
END OF FILE	Selects end-of-file status and signals card reader interface logic.
VALIDITY ON	Selects validity checking logic (Hollerith mode only).
NOT READY	Indicates that card reader is not ready to begin reading when indicator is lighted.
READ CHECK	Indicates that one of the photocell exciter lamps is not on or blocked.

NOTE

READ CHECK does not mean that a card was read incorrectly. This light cannot be energized while a card is actually passing through the read station. It can be set while no cards are in motion, before a card has entered the read station, or after it has left the read station.

FEED CHECK	Indicates a card jam or that a card has failed to feed (did not reach the read station).
VALIDITY CHECK	Indicates that a code other than a legitimate Hollerith code was read while in the Hollerith mode and with the VALIDITY ON switch energized. All illegal codes will be input to the computer as octal 00.

Placement of Cards in Hopper

The card reader accepts standard 7-3/8 x 3-1/4 in. punch cards 0.0070 in. thick. Cards should be placed in the hopper face down with the row-9 edge toward the back of the card reader.

Sample Program

The following subroutine is intended as an example only. When it is called, it will read Hollerith data from a card and pack it two characters per word for a maximum of 40 words. To keep the example simple, no provision has been made to convert from the 6-bit Hollerith code to the 8-bit ASCII code, nor was any error checking done.

```
        SUBR    CARD      Subroutine name
        REL
CARD    DAC     **        Subroutine entry name
        LDX     =-40      Set index for 80 character XFER
        SKS     '305      Test card reader operational
        JMP     *-1       Delay until operational
        SKS     '105      Test card reader busy
        JMP     *-1       Delay until not busy
        OCP     '5        Read one Hollerith card
STRT    INA     '1005     Clear A and Input character
        JMP     *-1       Delay if not ready
        ICR               Shift to pack
        INA     '5        Input character
        JMP     *-1       Delay if not ready
        STA*    BUF+40,1  Store word
        IRS     0         Increment index
        JMP     STRT      Loop to read next two characters
        JMP*    CARD      Return
BUF     BSS     40        Packed word storage
        END     CARD      End of subroutine
CARD    SUBR    CARD
```

SECTION IV
DAP-16 LANGUAGE

DAP-16 is a symbolic assembly program which translates a symbolic (source) program into machine language (object) code. DAP-16 provides symbolic programming while maintaining the characteristics, flexibility, speed and conciseness of machine language programming, and permits the assignment of symbolic addresses to storage locations.

MODES OF OPERATION

DAP-16 operates in two basic modes: load and desectorizing. When operating in the load mode DAP-16 closely approximates the addressing structure of the Honeywell 316/516 computer. Operand addresses must be within the same sector as the instruction, or in sector zero, otherwise an error flag is generated. This means that the programmer must be aware of an operand's location with respect to sector boundaries. Programs assembled in the LOAD mode are always absolute. It is the programmer's responsibility to provide all indirect linkage required for intersector addressing and subroutine calls.

In the desectorizing mode (Figure 4-1), DAP treats the Honeywell 316/516 class computer as if all of memory (up to 32K with Extended Addressing option for the DDP-516) is directly addressable. The desectorizing mode does not require the programmer to be concerned with the location of the operands with respect to sector boundaries. It also makes possible the writing of very efficient, completely relocatable programs. In the desectorizing mode, an extended object code is generated which provides the DAP/FORTRAN relocating loader with sufficient information to determine whether indirect address linkage must be supplied for any memory referencing instruction or whether the address may be inserted directly into the memory address instruction. Programs assembled in the desectorizing mode generally require less memory space and operate faster because the tedious chore of defining and minimizing indirect address links is done by DAP-16 and the loader rather than by the programmer. In the desectorizing mode, subroutines can be called using the CALL pseudo-operator, and all subroutine linkage is automatically completed by the DAP/FORTRAN relocating loader. Programs may be assembled in the desectorizing mode by placing an ABS pseudo-op (in the case of absolute programs) or a REL pseudo-op (in the case of relocatable programs) at the beginning of the program to be assembled. Programs written to be assembled in the desectorizing mode should not, in general, modify or move memory referencing instructions within the program during the course of program execution. The common practice of address modification may be easily and safely accomplished by making the address to be modified an indirect address link (using the DAC pseudo-operation). This indirect address link may then be modified in the desired manner.

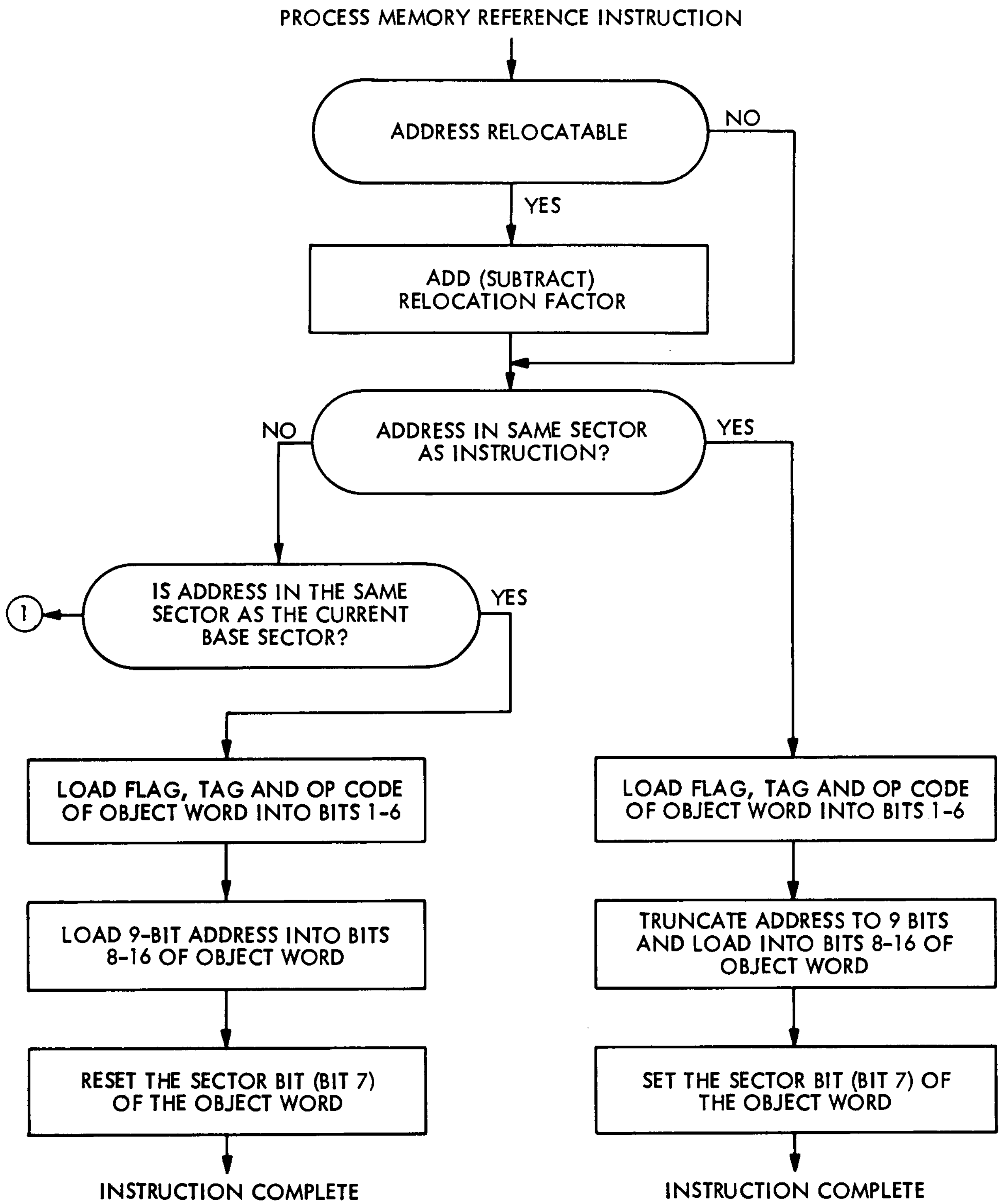


Figure 4-1. Desectorized Program Loading (Sheet 1 of 3)

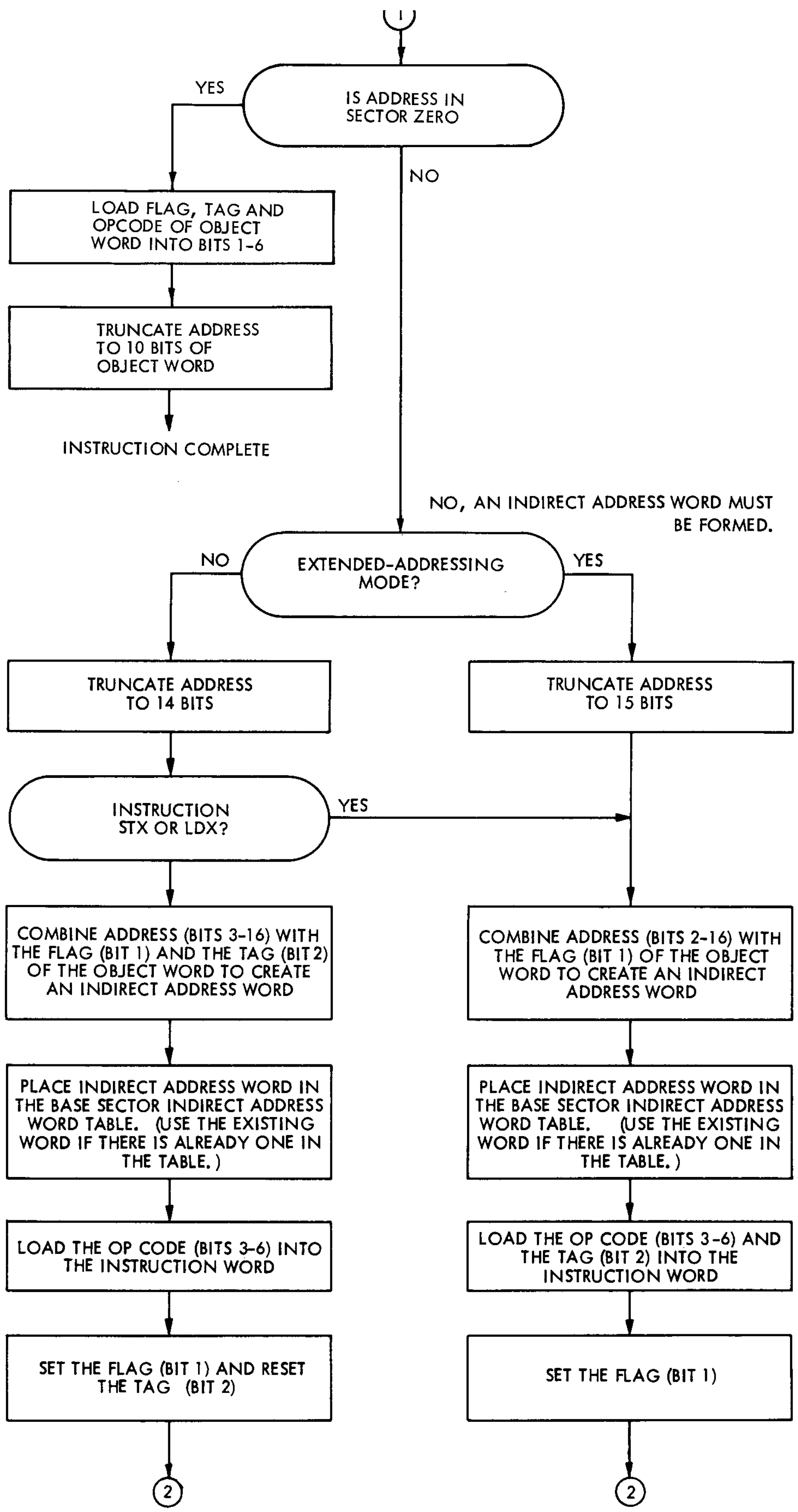


Figure 4-1. Desectorized Program Loading (Sheet 2 of 3)

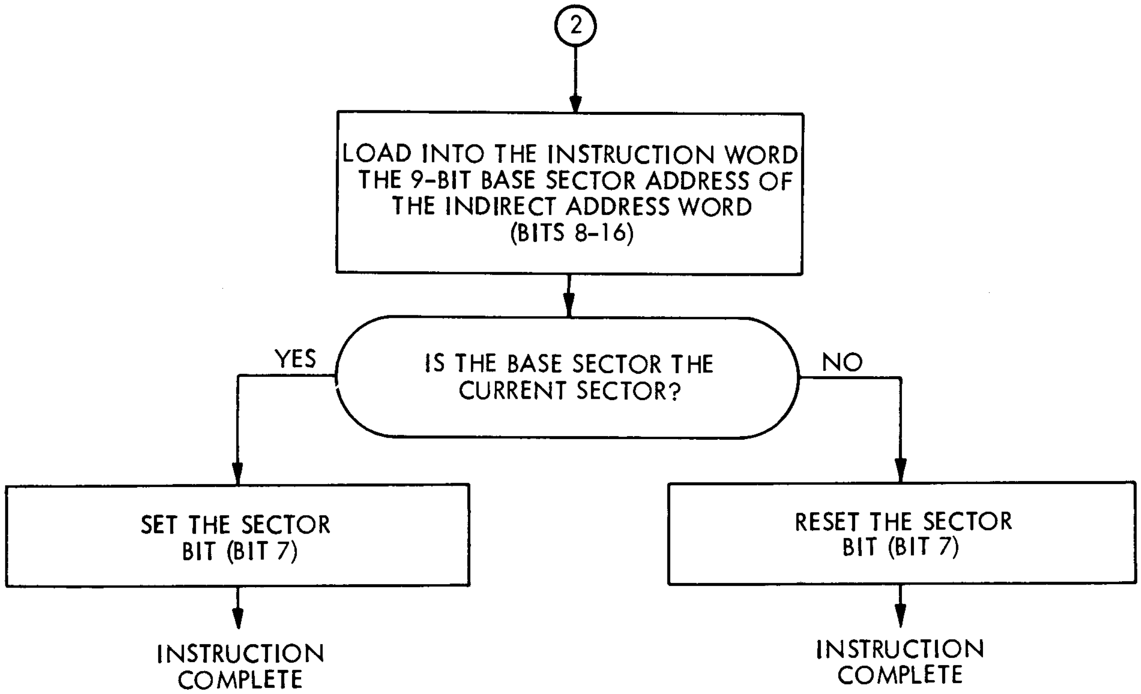


Figure 4-1. Desectorized Program Loading (Sheet 3 of 3)

ASSEMBLY PROCESS

Initially the DAP-16 assembler is loaded into computer memory. The sequence of symbolic instructions in the source program to be assembled are examined once or twice by DAP-16 at the programmers option. The contents of the A-register controls the number of passes and also the input/output device selection. If bit 1 (the sign bit) of the A-register is set to a 1, the two-pass mode is selected; if bit 1 is set to a 0, the one-pass mode is selected. The significance of the remaining 15 bits is discussed in Section V.

Two-Pass Assembly

The sequence of symbolic instructions in the source program to be assembled are examined twice by DAP-16: once to develop a dictionary of symbols, and a second time to assemble the object program by referencing the dictionary. The DAP-16 dictionary has storage space for defining operation mnemonics and symbols. Three cells are used for each operation or symbol; the encoded symbol or operation mnemonic is stored in the first two cells and defined in the third. For machine instructions, the definition cell contains the corresponding operation code. For location names, the definition cell contains the address at which the symbol is defined. For pseudo-operations, the definition cell contains a DAC to the location of the pseudo-operation analyzer in DAP-16. DAP-16 obtains locations for symbols by stepping a location counter for each line of the source program. The size of the symbol table may be calculated by the following formula:

$$\frac{\text{Top of memory - Highest location used - 100}}{3}$$

All calculations are in octal.

Program assembly takes place during pass two. Printing of each line is completed before the next line is started, reducing requirements for storage space. The line is read from the tape or card, stored in a special buffer (part of memory), the instruction or data word assembly is performed and, if requested, the assembled line is printed. Punching of the object program and punching or printing of the assembly listing are under control of the contents of the A-register.

Figure 4-2 illustrates how each line is processed. DAP-16 calls the subroutines necessary for reading and storing one line of type. The line is separated into its constituent fields, and the operation mnemonic is examined. The nature of the indicated operation (normal or pseudo) determines the subroutines to be called to process the operation field. For normal operations DAP-16 determines the specified machine operation by table look-up and then places the operation code in the appropriate portion of the instruction word being assembled. For pseudo-operations, analytical subroutines are called and serve to modify the assembly process, allocate storage, define data words, or provide for program linkages at load time.

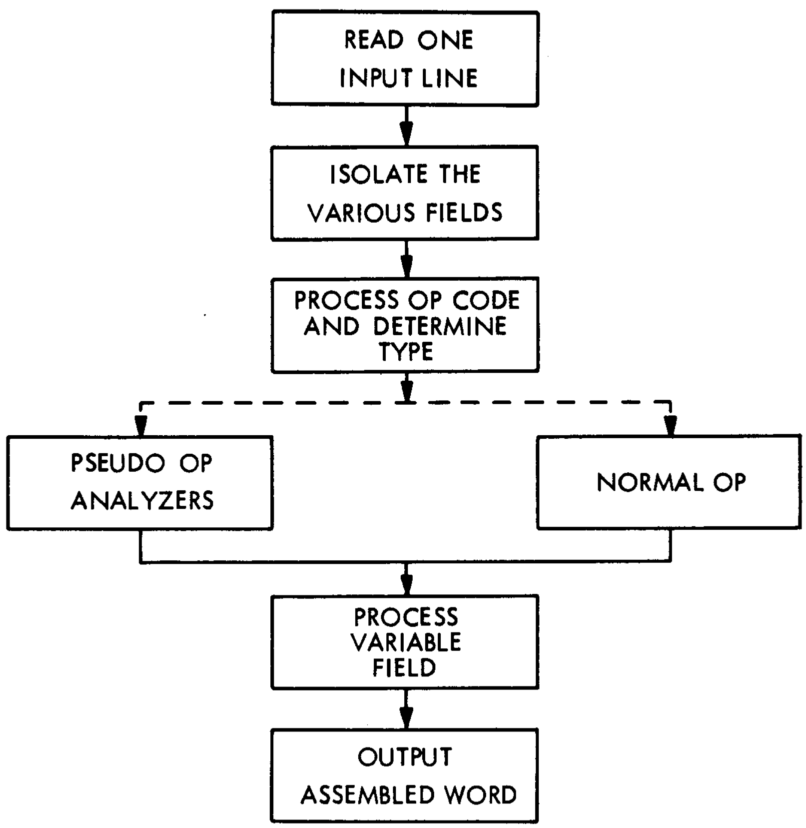


Figure 4-2. Processing of One Line

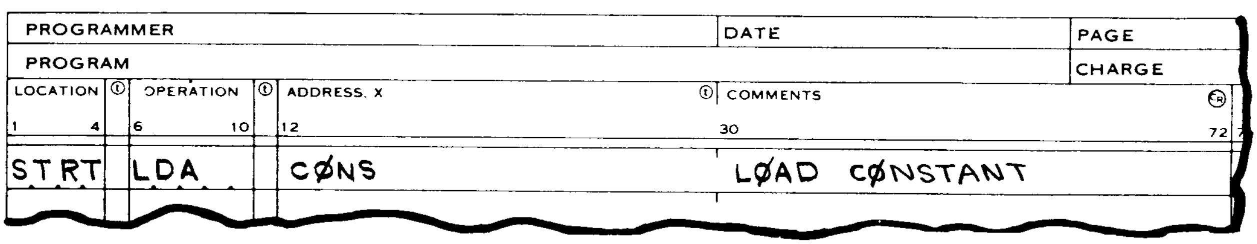
The variable field is then processed. Alphanumeric, octal, and decimal information is converted to binary; the DAP-16 dictionary is then searched to evaluate symbols and calculations are performed to evaluate expressions. If the operation field specified a normal machine operation, the resultant value forms the address field of the instruction being assembled.

One-Pass Assembly

The dictionary development and the object program assembly is accomplished in the same pass in a one-pass assembly. Forward referenced symbols (those that are used before being defined) have an unknown value at the assembly time. DAP-16 flags such symbols with a double asterisk (**) and assigns each symbol an internal symbol number which is outputted with the instruction in which the symbol occurs. The loader program maintains a table of symbol numbers and their use. When the value of the symbol becomes known, DAP-16 outputs the value along with the object program so that the loader can fill in references to the symbol. The object program resulting from a one-pass assembly is longer than that for a two-pass assembly because of the additional information that must be supplied to the loader. Programs assembled in the one-pass mode must be loaded by the extended version of the DAP/FORTRAN loader (LDR) rather than the standard loader (SLDR).

SOURCE LANGUAGE FORMAT

Programs written in the DAP-16 source language consist of a sequence of symbolic instructions or statements known as source lines. The example below shows a typical symbolic instruction written on a DAP-16 coding form. This instruction represents one source line.



As indicated in the coding sheet, symbolic instructions consist of four fields as follows:

a. The LOCATION field occupying character positions 1 through 4 of the source line.

b. The OPERATION field occupying character positions 6 through 10 of the source line.

c. The VARIABLE field beginning at character position 12 and continuing until a blank character or column 72 is present. This field is subdivided into the address and index subfields. The address and index subfields are separated by a comma.

d. The COMMENTS field begins at the character following the first blank character which terminates the VARIABLE field.

The above example shows an instruction which is located at the symbolic location STRT. The effect of the instruction is to load a constant, located at the symbolic location CONS, into the A-register. The comments field has no effect on the program. The significance of the several fields are discussed in more detail in the paragraphs that follow.

Location Field

The location field may be used to assign a symbolic address or "label" to an instruction so that the instruction can be referred to elsewhere in the program. The symbolic address in the location field consists of one to four characters, at least one of which is non-numeric. DAP-16 assigns memory addresses to the symbolic locations when assembling the object program.

Operation Field

The operation field is analogous to the operation-code portion of a machine language instruction. The contents of the operation field may be either a machine language instruction mnemonic or one of the pseudo-operation mnemonics in the DAP-16 repertoire. Operation mnemonics are either three or four characters in length. In addition to specifying an operation, the operation field may also specify that indirect addressing is desired by writing an asterisk (*) immediately following the operation-code mnemonic.

Variable Field

The variable field is normally used to specify an address and index register for DDP-16 class instructions. When used with a DAP-16 pseudo-operation, the significance of the variable field depends upon the nature of the pseudo-operation. (Pseudo-operations are discussed in Section V.)

Comments Field

The comments field may be used for any comments the programmer cares to write. This field has no effect on the assembler but it is printed out on the symbolic assembly listing. The format of the assembly listing is shown in Figure 4-3.

The portion of the assembly listing appearing on the right is a copy of the original source program input.

SYMBOLOGY

In addition to operation and pseudo-operation mnemonics, the DAP-16 language contains symbols, expressions, and literals. A number of rules, discussed below, govern the formation and usage of these language elements.

Error Field	Line Count	Program Location	Extended Machine Code (15-Bit Address)	Location	Op code	Variable	Card Sequence No.
	0001			*SAMPLE ASSEMBLY LISTING			0010
	0002				ORG	512	0020
	0003	01000	0 02 01001	STRT	LDA	*+1	0030
	0004	01001	0 04 01000		STA	*-1	0040
A	0005	01002	-0 02 00000		LDA*		0050
	0006	01003	0 06 01010		ADD	=15	0060
	0007	01004	0 06 01011		ADD	='15	0070
	0008	01005	0 04 00700		STA	STRT-64	0080
	0009	01006	0 02 01012		LDA	='-5	0090
	0010	01007	0414 76		LGL	2	0100
		01010	000017				0110
		01011	000015				0120
		01012	177773				0130
	0011				END	**	0140

Figure 4-3. Assembly Listing

Symbols

Symbols generally represent memory addresses and may appear in both the location and the variable fields of the symbolic instructions. The programmer defines a symbol by placing it in the location field of an instruction, thus giving the instruction a symbolic address. The assembly program keeps track of the location of instructions in the source program by stepping a location counter by one for each instruction. When a symbol appears in the location field it is normally assigned the current value of the location counter. The first such occurrence constitutes the definition of the symbol, and any subsequent occurrence in the location field causes an error printout. Undefined symbols, that is, symbols, appearing in the variable field of an instruction and not in any location field, cause an error printout. The value of an undefined symbol is some location at the end of the program.

Symbols consist of 1 to 4 characters from among the 37-character set of the letters of the alphabet, the 10 digits and the dollar sign character ($). At least one of the characters in any symbol must be alphabetic. The $ character should be used with care since it is used in column 1 by the update program to flag a command card.

The following symbols are legitimate.

LOOP
STP2

Expressions

Expressions appear only in the variable field and may be either simple (composed of a single element) or compound (composed of two or more elements separated by operators). An element may be either a symbol, a decimal integer less than or equal to 65535, an octal integer preceded by an apostrophe less than or equal to '177777, a single asterisk, or a double asterisk.

When a single asterisk appears in the variable field as an element, it designates an address equal to the current value of the location counter. Thus, * + 1 means "this location plus one." A double asterisk has a value of zero and is commonly placed in the variable field when the address is to be modified later by the program.

Operators are used to separate elements in compound expressions. An operator may be either a plus (addition) or a minus (subtraction). Only one operator is permissible between each pair of elements.

Expressions may have either relocatable or absolute modes. A relocatable expression is one that is relative to the first instruction of the program; an absolute expression is one which has a constant value regardless of its relative position in the program (e.g., an integer). The overall mode of the expression depends on the mode of each of the individual elements used to make up the expression.

Any permissible expression may be written to represent the address portion of a standard instruction. Additionally, the standard index (location zero) may be specified by following the address expression with a comma and the integer one.

The following are examples of valid expressions:

Assume (P) is '203	then	Q + 5 =	'12
Q = '5		ZZ + 2 =	'15
ZZ = '13		* =	'203
R = '20		* - Q =	'176
		* + 3 + Q - ** + '17303 - R =	'17476

Literals

Reference to a memory location containing a constant may be accomplished by use of one of the data defining pseudo-operations provided in the DAP-16 language. However, it is sometimes more convenient to represent a constant literally rather than symbolically. Consider the following example.

PROGRAMMER			DATE	PAGE
PROGRAM				CHARGE
LOCATION (1–4)	OPERATION (6–10)	ADDRESS, X (12)	COMMENTS (30)	(72)
	LDA	A		
A	DEC	50		

The first instruction refers to the symbolic constant A. The second instruction defines the constant as having the decimal value 50. An equivalent reference to the constant would have been as follows.

PROGRAMMER		DATE	PAGE
PROGRAM			CHARGE
LOCATION 1 4	OPERATION 6 10	ADDRESS, X 12	COMMENTS 30 72
	LDA	=50	

In this example, DAP-16 interprets the =50 as a decimal literal and automatically generates and assigns a location for the value 50. The resultant location of the value 50 is inserted into the address portion of the LDA instruction in the object program.

Three types of literals, decimal, octal, and ASCII are interpreted by DAP-16. A decimal literal consists of the equals character (=), followed by the sign (if no sign, the number is positive), followed by a fixed-point decimal integer. The rules for forming an octal literal are identical except that an apostrophe (') must follow the equals character. ASCII literals consist of the equals character followed by an A (=A), followed by two ASCII characters. If only one ASCII character is specified the second character is assumed to be a blank. The ASCII literals are an exception to the rule governing blanks in the variable field. The two characters following the "A" form the literal and the third character must be either a blank (end of the variable field) or a comma (beginning of the index subfield).

Asterisk Conventions

The conventions for use of the asterisk are summarized below.

a. An asterisk (*) in column 1 or first character in the location field: treat the entire card or line as remarks.

b. An asterisk (*) appended to instruction mnemonic: set the indirect address flag.

c. An asterisk (*) as an element: current value of the location counter.

d. A double asterisk (**) as a symbolic address: put zeros in address field (address is modified by another instruction).

e. A triple asterisk (***) as an operation code: op-code will be modified by another instruction. The instruction is assembled as a memory reference instruction with an operation code of 00_8.

ASSEMBLY LISTING

The printed output of DAP-16 is called the assembly listing. It is a printing of the symbolic input instructions in the order in which they appeared, together with the octal representation of the binary words produced by the assembler. A sample listing is shown in Figure 4-3. The first column contains the line ID number, which identifies the line and is used by the source-program update routine. The next column shows the memory location assigned to each instruction. The third column shows, in octal, the binary word assigned to the location.

The following observations taken from Figure 4-3 are intended to aid the reader in analyzing the characteristics of DAP-16.

a. Line 1 contains an asterisk in the location field, causing DAP-16 to treat the entire line as remarks.

b. Line 2 contains a pseudo-operation (ORG) which sets the DAP-16 location counter to octal 1000, the starting address of sector one.

c. The expression in the variable field in line 3 means the current value of the location counter, plus one. Consequently, DAP-16 has written octal 1001 into the address field of the instruction word assigned to this location.

d. The symbol in the left margin of line 5 is a diagnostic. Diagnostics are explained in Section V.

e. In line 10, the programmer has entered the number of shifts desired in an LGL instruction. DAP-16 has generated the necessary two's complement form in the object program.

f. Following line 10 is a literal pool of the three literals called for by the program.

SECTION V
DAP-16 PSEUDO-OPERATIONS

This section contains descriptions of all pseudo-operations provided in the DAP-16 language. Ancillary discussions of program relocation, data formatting, and program linkages are included to clarify pseudo-operation functions. (For a summary listing of DAP-16 pseudo-operations, see Appendix H.)

ASSEMBLY CONTROLLING PSEUDO-OPERATIONS

Assembly controlling pseudo-operations (ABS, CFx, END, FIN, LOAD, MOR, ORG, and REL) are used to start and stop program assembly and to select the assembly mode. Programs may be assembled in either the absolute or relocatable mode. Relocatable programs can be placed anywhere within memory at the time of loading, whereas absolute programs must be placed in their assembled locations.

During program assembly, DAP-16 maintains a location counter to assign memory locations for each data and instruction word that is assembled. The output of the location counter is shown on the assembly listing (Figure 4-3). If the program is assembled in the absolute mode, the DAP-16 loader loads the object program into the locations shown on the assembly listing. If the program is assembled in the relocatable mode (specified by an REL pseudo-operation), the loader loads the object program into the memory area specified by the programmer at program loading time. It is recommended that the main program be loaded at a starting address equal to or greater than 1000_8, so that sector zero can be used exclusively for address linkage and transfer vectors.

DAP-16, in the absence of a LOAD or REL pseudo-operation, assembles programs in the absolute mode. Relocatable programs are tentatively assembled for loading at a starting location of zero. However, at load time, a relocation constant is added to or subtracted from the address field of memory reference instructions and data words which reference symbolic locations. The relocation constant is equal to the difference between zero and the program starting location selected at load time.

When assembling relocatable programs, DAP-16 inserts control bits into the object program (not shown in the assembly listing) that enable the loader to identify instruction and data words referencing symbolic memory locations. The loader then adds the relocation constant to the address fields of these words.

ABS Pseudo-Operation

The ABS (absolute) pseudo-operation is used to direct DAP-16 to assemble subsequent instructions in the absolute mode. The contents of the symbolic instructions containing the ABS pseudo-operation are:

LOCATION	Ignored
OPERATION	ABS
VARIABLE	Ignored
COMMENTS	Normal

The effect of the ABS pseudo-operation is to assign absolute locations to the instructions assembled. The assembler continues to run in the absolute mode until a REL, LOAD, or END pseudo-operation is encountered. The ABS mode is the normal assembly mode.

CFx Pseudo-Operation

The CFx (configuration) pseudo-operation is used to inform DAP-16 as to which DDP-16 class computer the object program is to be executed on. The suffix "x" has the following connotation: 1 for the H116, 3 for the H316, 4 for the H416, and 5 for the DDP-516. If the configuration is not specified it is assumed that the object program is to be executed on the same Series H-16 class computer as that on which the assembly is being performed. The contents of symbolic instructions containing the CFx pseudo-operation are:

LOCATION	Ignored
OPERATION	CF1, CF3, CF4, or CF5
VARIABLE	Ignored
COMMENTS	Normal

The CFx pseudo-operation causes the DAP-16 to flag any instructions that are illegal for the object computer without interrupting the assembly.

END Pseudo-Operation

The END pseudo-operation is used to direct DAP-16 to terminate the current assembly pass and prepare for the second pass if the two-pass mode has been selected. The contents of symbolic instructions containing the END pseudo-operation are:

LOCATION	Ignored
OPERATION	END
VARIABLE	(1) An expression that defines the address of the instruction to which control should be transferred at the conclusion of the loading process at object time. If the variable field is left blank, the transfer address is set to the location of the first instruction in the main program. (2) Subroutine; ignored.
COMMENTS	Normal

The END pseudo-operation causes DAP-16 to perform the following functions:

a. The current block of assembly output information is terminated.

b. All literals are punched out and undefined symbols are assigned locations.

c. An end jump block is punched following the assembly output. The jump address is the value of the expression in the variable field. If the variable field is left blank, the transfer address is set to the first instruction in the main program.

d. The assembly process is terminated if the current pass is the final one.

The END pseudo-operation must be the last statement in the source program.

When operating in the two-pass mode, the START pushbutton must be depressed to start processing pass two. While the computer is halted, the operator must reposition the source tape to the beginning, or reload the card deck.

FIN Pseudo-Operation

The FIN (finish) pseudo-operation is used to direct DAP-16 to punch out all literals accumulated up to the point at which the FIN pseudo-operation is initiated. The contents of symbolic instructions containing the FIN pseudo-operation are:

LOCATION	Ignored
OPERATION	FIN
VARIABLE	Ignored
COMMENTS	Normal

The effect of the FIN pseudo-operation is to cause DAP-16 to punch out all the accumulated literals. The purpose of this pseudo-operation is to permit literals to be interspersed throughout the program thus minimizing the necessity for indirect address links when referencing literals.

LOAD Pseudo-Operation

The LOAD pseudo-operation is used to direct DAP-16 to flag any instruction address that required desectorizing. The contents of symbolic instructions containing the LOAD pseudo-operation are:

LOCATION	Ignored
OPERATION	LOAD
VARIABLE	Ignored
COMMENTS	Normal

The effect of the LOAD pseudo-operation is to cause DAP-16 to flag any instruction whose address refers to a location outside the current sector or zero. The assembler continues to operate in the LOAD mode until an END, REL, or ABS pseudo-operation is encountered.

MOR Pseudo-Operation

The MOR (more) pseudo-operation causes the computer to halt and await operator action (except when magnetic tape input has been selected in which case MOR is ignored). The contents of symbolic instructions containing the MOR pseudo-operation are:

LOCATION	Ignored
OPERATION	MOR
VARIABLE	Ignored
COMMENTS	Normal

ORG Pseudo-Operation

The ORG (origin) pseudo-operation sets the location counter to a specified value. The contents of symbolic instructions containing the ORG pseudo-operation are:

LOCATION	Normal
OPERATION	ORG
VARIABLE	Normal. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The ORG pseudo-operation performs the following functions:

a. The expression in the variable field is evaluated.

b. The location counter is set to the value thus determined.

A symbol in the location field of an ORG pseudo-operation is assigned the value of the location counter prior to processing the ORG pseudo-operation. Consider the following example.

PROGRAMMER | DATE | PAGE
PROGRAM | CHARGE

LOCATION 1–4	OPERATION 6–10	ADDRESS, X 12	COMMENTS 30–72
	ØRG	'100	
	NØP		
FUNA	ØRG	'1000	
	LDA	X	

The LDA instruction is assigned to an absolute location (1000_8). The symbol FUNA is assigned the absolute value 101_8.

REL Pseudo-Operation

The REL (relocatable) pseudo-operation is used to direct DAP-16 to assemble the subsequent instructions in the relocatable mode. The contents of symbolic instructions containing the REL pseudo-operation are:

LOCATION	Ignored
OPERATION	REL
VARIABLE	Ignored
COMMENTS	Normal

The effect of the REL pseudo-operation is to cause DAP-16 to assign relative locations to the instructions assembled. The assembler continues to run in the relocatable mode until the END pseudo-operation is encountered or until an ABS or a LOAD pseudo-operation is encountered.

DATA DEFINING PSEUDO-OPERATIONS

The data defining pseudo-operations (BCI, DAC, DBP, DEC, and OCT) are used for defining constants and generating data for inclusion in the object program. The operations in this category cause DAP-16 to interpret alphanumeric data, decimal numbers, and octal numbers, respectively. The somewhat complex rules and restrictions for forming expressions in the variable field in the DEC pseudo-operation are discussed in the paragraphs immediately following the summary coverage of format and content.

Decimal and octal constants also can be generated by the use of literals as discussed in Section IV.

BCI Pseudo-Operation

The BCI (binary coded information) pseudo-operation is used to direct DAP-16 to generate binary words in ASCII form from alphanumeric data. The contents of symbolic instructions containing the BCI pseudo-operation are:

LOCATION	Normal
OPERATION	BCI
VARIABLE	N, followed by 2N alphanumeric characters. The N specifies the number of words to be converted and may not exceed 29.
COMMENTS	Normal

The effect of the BCI pseudo-operation is to convert each group of two characters into a left-justified binary word in ASCII code. These words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field it is assigned the same location as the first word of binary data generated by the pseudo-operation. The alphanumeric characters in the message to be encoded must be counted and entered as the first subfield. A typical example is shown below (six words of storage required). The BCI pseudo-operation is an exception to the rule in that the first blank terminates the variable field. The comments field begins immediately following the last character included in the character count.

PROGRAMMER		DATE	PAGE
PROGRAM			CHARGE
LOCATION 1–4	OPERATION 6–10	ADDRESS, X 12	COMMENTS 30–72
FINI	BCI	6,REMOUNT TAPE	

DAC Pseudo-Operation

The DAC (define address constant) pseudo-operation directs DAP-16 to generate a 16-bit binary word which can be used by flagged memory reference instructions to access an operand in any memory sector. The contents of symbolic instructions containing the DAC pseudo-operation are:

LOCATION	Normal
OPERATION	DAC or DAC*
VARIABLE	Normal
COMMENTS	Normal

The DAC pseudo-operation causes DAP-16 to evaluate the expression in the variable field and assemble a 16-bit address word. When the flag or tag (bit 1 or 2) is specified as part of the address word, the value of constant generated is increased by 100000_8 or 40000_8, respectively. It is the programmer's responsibility to ensure that addresses over 37777 are not mistaken for flags and tags and vice-versa.

DEC Pseudo-Operation

The DEC (decimal) pseudo-operation is used to direct DAP-16 to generate binary words from decimal data. The contents of symbolic instructions containing the DEC pseudo-operation are:

LOCATION	Normal
OPERATION	DEC
VARIABLE	One or more subfields, each containing a decimal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters in the instruction line must not exceed 72. Rules for forming the decimal subfields are discussed below.
COMMENTS	Normal

The effect of the DEC pseudo-operation is to cause DAP-16 to convert each subfield to one, two, or three binary words, depending on whether the decimal data is single precision fixed-point, double precision fixed-point, single precision floating-point, or double precision floating-point. These words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field it is assigned the same location as the first word of binary data generated by the pseudo-operation.

Fixed-Point Decimal Data. -- Fixed-point decimal data may be either single precision or double precision. A significance of four decimal digits can be maintained in single precision, fixed-point arithmetic on the DDP-516. In many arithmetic operations, this degree of significance is adequate and is desirable because of the enhanced speed of computation. A single precision fixed-point decimal number requires one computer word (sign and 15 bits of significance) and is written in two parts: the significant part and the scaling part. Double precision fixed-point data consists of two words (sign and 30 significant bits).

The significant part of the fixed-point number is a signed or unsigned decimal number with or without a decimal point. If the decimal point is not specified, it is assumed to be immediately to the right of the last digit (a decimal integer).

The scaling part of the fixed-point number is the letter B (for single precision) or the letters BB (for double precision), followed by a signed or unsigned decimal integer specifying the position of the understood binary point. If the scaling part is not present, the number is interpreted as a truncated decimal integer whose understood binary point is immediately to the right of the LSB in the computer word (position 16).

The general form of the scaling part is B ±NN or BB ±NN, where NN gives the position of the understood binary point relative to the machine binary point. The minus sign defines the understood binary point to be to the left of the machine binary point, and the plus (or no sign) defines the understood binary point to be to the right of the machine binary point. The machine binary point is defined to be between the sign bit and the most significant bit of the computer word; i. e., between bit positions 1 and 2.

In addition to a scaling part, fixed-point numbers may also have an exponent part specified by the use of an E field in addition to a B field. E fields are discussed more fully in paragraphs on floating-point data.

The examples below show how DAP-16 produces fixed-point numbers. The left column shows the decimal number to be translated. This is written in the variable field. The right column shows the resultant octal word that would be generated by DAP-16. Single precision fixed-point numbers are limited to magnitudes less than 2^{15}.

15	000017
15B+15	000017
15.001B5	036001
15.001BB5	036001
	003044
-.002B-2	177372

Floating-Point Decimal Data. -- Floating-point data may be either single or double precision. A single precision, floating-point number requires two computer words (sign, 8-bit characteristic, and 23-bit fraction). A double precision, floating-point number requires three computer words (sign, 8-bit characteristic, and 39-bit fraction).

A decimal floating-point number is written as two parts: the significant part and the exponent part. The significant part of a floating-point number is a signed or unsigned decimal number written with a decimal point.

The exponent part of the decimal floating-point number is the letter E or the letters EE followed by a signed or unsigned decimal integer. The exponent part serves the following purposes.

a. It indicates whether the floating-point number is to be single (E) or double precision (EE).

b. It specifies a constant in the form of 10 raised to the indicated power by which the significant part of the number is to be multiplied.

The resulting 8-bit binary exponent is expressed in 128 excess arithmetic and allows for numbers in the range of $10^{\pm 38}$.

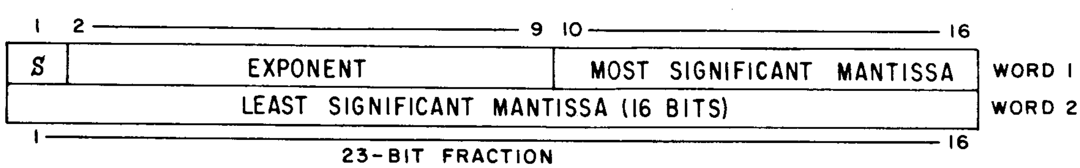
All negative floating-point numbers are expressed in two's complement form, which means that the exponent in this case is in one's complement form.

Figure 5-1 shows the formats of floating-point numbers and Table 5-1 shows various examples of floating-point numbers generated by the DEC pseudo-operation. The left column shows the decimal number to be translated and the right column shows the octal words that would be generated by the DEC pseudo-operation. The fractional portion of the floating-point number is always normalized by DAP-16.

DBP Pseudo-Operation

The DBP (double precision) pseudo-operation directs DAP-16, when assembling on an H316/516 with the double precision option, to generate binary words from decimal data. The contents of symbolic instruction containing the DBP pseudo-operation are:

LOCATION	Normal
OPERATION	DBP



A. Single-Precision Format

1	2 ——— 9	10 ——— 16	
S	EXPONENT	MOST SIGNIFICANT MANTISSA	WORD 1
NEXT MOST SIGNIFICANT MANTISSA (16 BITS)			WORD 2
LEAST SIGNIFICANT MANTISSA (16 BITS)			WORD 3

39 - BIT FRACTION

B. Double-Precision Format

Figure 5-1. Floating-Point Formats

Table 5-1.
Floating-Point Number Translations

Decimal Number	Octal Translation	Remarks
.15E2	041170 000000	.15 times 10^2 = 15
+.15E + 2	041170 000000	Same as first example
-.15E2	136610 000000	Negative of first example
1234E-5	036545 013333	Expression = .01234
.123	037375 171666	Single-precision
.1E0	037346 063146	Single-precision; binary exponent is negative
.1EE0	037346 063146 063146	Double-precision result

VARIABLE — One or more subfields, each containing a decimal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters in the instruction line must not exceed 72.

The effect of the DBP pseudo-operation is the same as that of the DEC pseudo-operation with the exception that the DBP always loads an even location and always generates a double precision constant.

OCT Pseudo-Operation

The OCT (octal) pseudo-operation directs DAP-16 to generate binary words from octal data. The contents of symbolic instructions containing the OCT pseudo-operation are:

LOCATION	Normal
OPERATION	OCT
VARIABLE	One or more subfields, each containing an octal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters on the instruction line must be limited to 72.
COMMENTS	Normal

The effect of the OCT pseudo-operation is to cause DAP-16 to convert each subfield to a binary word. The octal data entries are right-justified, and assigned to successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it is assigned to the same location as the first word of binary data generated by the pseudo-operation.

The only allowable characters in an octal field are: plus, minus, apostrophe, 0, 1, 2, 3, 4, 5, 6, 7, and commas separating the subfields. Octal numbers may be signed (limited to magnitudes less than 2^{15}) or unsigned (limited to magnitudes less than 2^{16}). If an octal number is unsigned, it is assumed to be positive. The appearance of an apostrophe preceding the octal number is acceptable but is redundant.

LOADER-CONTROLLING PSEUDO-OPERATIONS

The loader-controlling pseudo-operations (EXD, LXD and SETB) are used to enter or leave the extended addressing mode for desectorizing and to designate a memory sector other than sector zero as the base sector for cross sector linkage. Pseudo-operations EXD and LXD are valid only for those DDP-516 computers equipped with the Extended Memory option. Pseudo-operation SETB is valid primarily for those DDP-516 computers equipped with the Memory Lockout option. Programs containing the EXD, LXD or SETB pseudo-operations must be loaded using the extended DAP/FORTRAN loader (LDR) rather than the standard loader (SLDR).

EXD Pseudo-Operation

The EXD (enter extend-mode desectorizing) pseudo-operation directs the loader to desectorize the subsequent instructions for execution in the extended addressing mode. The contents of symbolic instructions containing the EXD pseudo-operation are:

LOCATION	Ignored
OPERATION	EXD
VARIABLE	Ignored
COMMENTS	Normal

The effect of the EXD pseudo-operation is to increase the size of loader-created indirect address words to 15 bits to increase addressing capability to 32K. This limits the extend mode to one level of indexing since the tag of the instruction word is not moved

LOCATION	Normal
OPERATION	OCT
VARIABLE	One or more subfields, each containing an octal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters on the instruction line must be limited to 72.
COMMENTS	Normal

The effect of the OCT pseudo-operation is to cause DAP-16 to convert each subfield to a binary word. The octal data entries are right-justified, and assigned to successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it is assigned to the same location as the first word of binary data generated by the pseudo-operation.

The only allowable characters in an octal field are: plus, minus, apostrophe, 0, 1, 2, 3, 4, 5, 6, 7, and commas separating the subfields. Octal numbers may be signed (limited to magnitudes less than 2^{15}) or unsigned (limited to magnitudes less than 2^{16}). If an octal number is unsigned, it is assumed to be positive. The appearance of an apostrophe preceding the octal number is acceptable but is redundant.

LOADER-CONTROLLING PSEUDO-OPERATIONS

The loader-controlling pseudo-operations (EXD, LXD and SETB) are used to enter or leave the extended addressing mode for desectorizing and to designate a memory sector other than sector zero as the base sector for cross sector linkage. Pseudo-operations EXD and LXD are valid only for those DDP-516 computers equipped with the Extended Memory option. Pseudo-operation SETB is valid primarily for those DDP-516 computers equipped with the Memory Lockout option. Programs containing the EXD, LXD or SETB pseudo-operations must be loaded using the extended DAP/FORTRAN loader (LDR) rather than the standard loader (SLDR).

EXD Pseudo-Operation

The EXD (enter extend-mode desectorizing) pseudo-operation directs the loader to desectorize the subsequent instructions for execution in the extended addressing mode. The contents of symbolic instructions containing the EXD pseudo-operation are:

LOCATION	Ignored
OPERATION	EXD
VARIABLE	Ignored
COMMENTS	Normal

The effect of the EXD pseudo-operation is to increase the size of loader-created indirect address words to 15 bits to increase addressing capability to 32K. This limits the extend mode to one level of indexing since the tag of the instruction word is not moved into the indirect address word. Therefore, bit 2 of the indirect address word is no longer interpreted as a tag but as part of the address.

LXD Pseudo-Operation

The LXD (leave extend-mode desectorizing) pseudo-operation directs the loader to desectorize subsequent instructions for execution in the normal addressing mode. The contents of symbolic instructions containing the LXD pseudo-operation are:

LOCATION	Ignored
OPERATION	LXD
VARIABLE	Ignored
COMMENTS	Normal

The effect of the LXD pseudo-operation is to restore loading to the normal addressing mode.

SETB Pseudo-Operation

The SETB (set base sector) pseudo-operation notifies the loader that a base sector other than sector zero will be used to execute subsequent instructions. The contents of symbolic instructions containing the SETB pseudo-operation are:

LOCATION	Normal
OPERATION	SETB
VARIABLE	Normal. Any symbol used in this field must have previously been defined.
COMMENTS	Normal

The pseudo-operation SETB designates the sector in which the indirect address words for cross sector linkage are to be stored. The value of the variable field designates the first location into which indirect address words are to be stored. Successive words are stored in successive locations. If a symbol appears in the location field, it is assigned the current value of the location counter.

The SETB pseudo-operation does not reserve a block of storage for the indirect address word table. It is the programmer's responsibility to reserve a block for the table in the proper place via a BSS pseudo-operation.

LIST-CONTROLLING PSEUDO-OPERATIONS

The list-controlling pseudo-operations (EJCT, LIST, and NLST) are used to control the printout of the source and object program assembly listing. These operations have no effect on the object program.

EJCT Pseudo-Operation

The EJCT (eject) pseudo-operation directs DAP-16 to begin or resume listing on a new page. The contents of symbolic instructions continuing the EJCT pseudo-operation are:

LOCATION	Ignored
OPERATION	EJCT
VARIABLE	Ignored
COMMENTS	Normal

The effect of the EJCT pseudo-operation is to cause the I/O selector program (IOS) to generate the necessary commands to advance the listing one page and continue listing on a new page. This pseudo-operation is valid only with systems having a line printer and is ignored if the pseudo-operation NLST is currently in effect.

LIST Pseudo-Operation

The LIST (listing) pseudo-operation directs DAP-16 to print a side-by-side listing of the program being assembled. The contents of symbolic instructions containing the LIST pseudo-operation are:

LOCATION	Ignored
OPERATION	LIST
VARIABLE	Ignored
COMMENTS	Normal

The effect of the LIST pseudo-operation is to cause the source program and its octal representation to be listed on the on-line typewriter or printer. The assembler then continues to operate in the listing mode until an NLST pseudo-operation is encountered. The assembler is normally in the LIST mode.

NLST Pseudo-Operation

The NLST (no listing) pseudo-operation directs DAP-16 to refrain from producing a side-by-side listing of the program being assembled. The contents of symbolic instructions containing the NLST pseudo-operation are:

LOCATION	Ignored
OPERATION	NLST
VARIABLE	Ignored
COMMENTS	Normal

The effect of the NLST pseudo-operation is to inhibit DAP-16 from listing the source program and its octal representation on the on-line typewriter or printer. The assembler then continues to operate in the no-listing mode until a LIST pseudo-operation is encountered. Initialization of the assembler automatically sets the listing mode.

PROGRAM LINKING PSEUDO-OPERATIONS

The DAP-16 pseudo-operations CALL and SUBR are used to generate communication links between programs. The CALL pseudo-operation initiates transfer of control to an external subroutine. The SUBR pseudo-operation defines points of entry into the subroutine from an external program.

The variable field of the CALL pseudo-operation contains the name of the external subroutine being called. Each time a particular subroutine is called, DAP-16 punches the subroutine name as a special block and assembles a JST (jump and store) operation to location zero. Then, as the object program is loaded into memory, the loader completes the program linkage by requesting and loading the external subroutine being called and filling in the address of the JST instruction, desectorizing it if necessary.

CALL Pseudo-Operation

The CALL (call) pseudo-operation directs DAP-16 to generate instructions that transfer control to a specified subroutine. The contents of symbolic instructions containing the CALL pseudo-operation are:

LOCATION	Normal
OPERATION	CALL
VARIABLE	A subroutine name (one to six characters)
COMMENTS	Normal

The effects of the CALL pseudo-operation are:

a. The subroutine name from the variable field is punched as a special block type.

b. A JST with an address of zero is entered into the sequence of assembled instructions.

c. If there is a symbol in the location field it is assigned to the location of the JST instruction inserted in step b.

XAC Pseudo-Operation

The XAC (external address constant) pseudo-operation directs the loader to generate a 16-bit binary word which is used by flagged memory reference instructions to access an operand outside the program. The contents of symbolic instructions containing the XAC pseudo-operation are:

LOCATION	Normal
OPERATION	XAC or XAC*
VARIABLE	External subroutine name (one to six characters) optionally tagged
COMMENTS	Normal

The XAC pseudo-operation causes the loader to evaluate the term in the variable field and assemble information which specifies that a reference is made outside the program.

The external location must be defined either in the current or a separate program assembly by SUBR pseudo-operation. At load time, after the external reference is defined, the true address, the flag, and the tag are generated and stored at the location of the XAC word.

SUBR Pseudo-Operation

The SUBR (subroutine) pseudo-operation is used to define a DAP-16 subroutine, and to symbolically assign a name to the subroutine for external reference.

The contents of symbolic instructions containing the SUBR pseudo-operation are:

LOCATION	Ignored
OPERATION	SUBR
VARIABLE	A one to six character name identifying an entry point to a subroutine optionally followed by a comma and a one to four character name defining the entry point. The name defining the entry point need be included only if it differs from the first four characters of the identifying name.
COMMENTS	Normal

The effect of the SUBR pseudo-operation is to cause the identifying name in the variable field to be generated in the object program output as identification for the loader. There must be as many SUBR pseudo-operations in a subroutine as there are entry points; however, the entry points may be multiply defined. The SUBR pseudo-operation must be the first operation of the subroutine, preceded only by another SUBR, if present.

The following is an example of a subroutine for which entry and return provisions have been made.

PROGRAMMER | DATE | PAGE

PROGRAM | CHARGE

LOCATION 1–4	OPERATION 6–10	ADDRESS, X 12	COMMENTS 30–72
	SUBR	SINE	
	REL		
SINE	DAC	**	START OF SINE ROUTINE
	JMP*	SINE	EXIT FROM SINE ROUTINE

Access to this subroutine from an external program is possible by use of the following instruction.

PROGRAMMER			DATE	PAGE
PROGRAM				CHARGE
LOCATION 1 4	OPERATION 6 10	ADDRESS, X 12	COMMENTS 30 72	
	CALL	SINE		

The following subroutine has two entry points and each entry point is defined twice.

PROGRAMMER			DATE	PAGE
PROGRAM				CHARGE
LOCATION 1 4	OPERATION 6 10	ADDRESS, X 12	COMMENTS 30 72	
	SUBR	SINE	NAME FOR SINE ROUTINE	
	SUBR	CØSINE	NAME FOR COSINE ROUTINE	
	SUBR	ARCTAN, ATAN	NAME FOR ARCTAN ROUTINE	
	SUBR	SINF, SINE	ALTERNATE NAME FOR SINE ROUTINE	
	REL			
SINE	DAC	**	START OF SINE ROUTINE	
	|			
	|			
	JMP*	SINE	EXIT FROM SINE ROUTINE	
CØSI	DAC	**	START OF COSINE ROUTINE	
	|			
	|			
	JMP*	CØSI	EXIT FROM COSINE ROUTINE	
ATAN	DAC	**	START OF ARCTAN ROUTINE	
	|			
	|			
	JMP*	ATAN	EXIT FROM ARCTAN ROUTINE	

Entry to the sine portion of the subroutine is made by

	CALL	SINE
or	CALL	SINF

Entry to the cosine portion of the subroutine is made by

CALL COSINE

Entry to the arc tangent portion of the subroutine is made by

CALL ARCTAN

Programs coded as subroutines (i. e., programs preceded by the SUBR pseudo-operation) cannot be loaded independently by means of the DAP-16 loader but must be called by a main program.

STORAGE ALLOCATION PSEUDO-OPERATIONS

The DAP-16 pseudo-operations (BES, BSS, BSZ, and COMN) enable the programmer to allocate memory cells for data storage or working space. For example, if a group of 350 integers are to be ordered and assembled in a table, the symbolic instruction shown below allocates 350 consecutive cells for storage of the integers in symbolic locations TABL through TABL + 349.

PROGRAMMER		DATE	PAGE
PROGRAM			CHARGE

LOCATION (1–4)	OPERATION (6–10)	ADDRESS, X (12)	COMMENTS (30–72)
TABL	BSS	350	

BES Pseudo-Operation

The BES (block ending with symbol) pseudo-operation is used for reserving storage locations. The contents of symbolic instructions containing the BES pseudo-operation are:

LOCATION	Normal
OPERATION	BES
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the BES pseudo-operation is to increase the value of the location counter by the value of the expression in the variable field. If there is a symbol in the location field it is assigned the value of the location counter <u>after</u> the increase. Consider the following example.

PROGRAMMER			DATE	PAGE
PROGRAM				CHARGE
LOCATION 1–4	OPERATION 6–10	ADDRESS, X 12	COMMENTS 30–72	
A	ØCT	5		
BLK	BES	5		
B	ØCT	6		

If A has been assigned location 50, BLK is assigned location 56, leaving five vacant cells; B is also assigned to location 56.

BSS Pseudo-Operation

The BSS (block starting with symbol) pseudo-operation is used for reserving storage locations. The contents of symbolic instructions containing the BSS pseudo-operation are:

LOCATION	Normal
OPERATION	BSS
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the BSS pseudo-operation is to increase the value of the location counter by the value of the expression in the variable field. If there is a symbol in the location field, it is assigned the value of the location counter before the increase. Consider the following example.

PROGRAMMER			DATE	PAGE
PROGRAM				CHARGE
LOCATION 1–4	OPERATION 6–10	ADDRESS, X 12	COMMENTS 30–72	
A	ØCT	5		
BLK	BSS	5		
B	ØCT	6		

In this case, if A has been assigned location 50, BLK is assigned location 51 and B is assigned location 56, leaving five vacant cells.

The BES and BSS pseudo-operations effect the punched output during assembly. When DAP-16 encounters one of these pseudo-operations, the block of machine instructions being accumulated in a special punch buffer (internal to DAP-16) is punched out, regardless of the number of words that have been accumulated. For BES and BSS, a new block is started with an origin address equal to the DAP-16 location counter after processing the BES or BSS pseudo-operation.

BSZ Pseudo-Operation

The BSZ (block storage of zeros) pseudo-operation is used for reserving storage locations that are initially (at load time) set to zeros. The contents of symbolic instructions containing the BSZ pseudo-operation are:

LOCATION	Normal
OPERATION	BSZ
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the BSZ pseudo-operation is to increase the value of the location counter by the value of the expression in the variable field. If there is a symbol in the location field it is assigned the value of the location counter before the increase.

COMN Pseudo-Operation

The COMN (common) pseudo-operation is used for assigning absolute storage locations in upper memory. The contents of symbolic instructions containing the COMN pseudo-operation are:

LOCATION	Normal
OPERATION	COMN
VARIABLE	Any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The effect of the COMN pseudo-operation is to cause DAP-16 to subtract the value of the expression in the variable field from the COMMON base and assign this value to the symbol in the location field. COMMON base is a user option. The COMN pseudo-operation establishes a common data pool that can be referenced by several programs.

SYMBOL DEFINING PSEUDO-OPERATION

A symbol defining pseudo-operation (EQU) is provided for assigning an absolute or relocatable value to a symbol.

EQU Pseudo-Operation

The EQU (equals) pseudo-operation is used for defining a value for a symbol for reference by other DAP-16 operations. The contents of symbolic instructions containing EQU pseudo-operation are:

LOCATION	Normal; must contain a symbol
OPERATION	EQU
VARIABLE	Any absolute or relocatable expression. Any symbol used in this field must have been previously defined.
COMMENTS	Normal

The EQU pseudo-operation causes DAP-16 to evaluate the variable field expression for value and to assign the value to the symbol in this location field. The mode of the symbol in the location field is the same as the mode of the expression in the variable field.

SPECIAL MNEMONIC CODES

Two special mnemonic codes are provided for the convenience of the programmer when writing special instruction groups for calling sequences. The mnemonic codes are assembled like any machine language instruction in that they may have address, index, and indirect fields. These codes are desectorized by the loader as 9-bit address memory reference instructions.

Mnemonic	Assembles As
PZE	Zeros in op-code
***	Zeros in op-code

OBJECT PROGRAM PREPARATION

Object program preparation consists of reading DAP-16 into computer memory then reading the source tape or card deck with the contents of the A-register set to provide the desired punching and printing options. Table 5-2 shows the significance of the various bit positions on both standard systems and those systems equipped with standard options. Principal options provided by DAP-16 are:

a. Punching the object program.
b. Punching or printing the assembly listing.
c. Punching the object program and printing the assembly listing simultaneously.
d. Assembling multi-section programs.

For very brief programs, option c provides an assembly listing for reference and, simultaneously, an object program for execution. When an assembly listing is desired for programs of normal length and a high-speed paper tape punch is available, the option of punching the assembly listing is most useful. The printed assembly listing can then be prepared off-line. Option d is useful for assembling programs prepared in several sections by use of the MOR pseudo-operation.

Table 5-2.
A-Register Bit Settings for I/O Device Selection

Bit	Meaning	Selection
1	1 for 2 pass, 0 for 1 pass	
2	Teletype	Source Device
3	Paper Tape Reader	
4	Card Reader	
5	Magnetic Tape No. 1	
6	Teletype with program halts provided for manual inputs	
*	No bits set. Source input from Disc.	
7	Teletype	Object Device
8	Paper Tape Punch	
9	Card Punch	
10	Magnetic Tape No. 2	
11	No Object Output	
*	No bits set. Object output to Disc.	
12	Teletype	List Device
13	Paper Tape Punch	
14	Magnetic Tape No. 3	
15	Line Printer	
16	No Listing	
*	No bits set. Listing output to Disc.	

* Only when used with DOP or DOP-S.

ERROR DIAGNOSIS

DAP-16 is able to detect many types of clerical errors commonly made in coding programs. These errors are indicated by an appropriate error code printed in the left margin of the assembly listing. (See Figure 4-3.) Examples of errors that are detected and their associated flags are as follows.

Error	Flag
Multiply defined symbol	M
Erroneous conversion of a constant or a variable field in improper format	C
Address field missing where normally required, or error in address format	A
Operation code missing or in error	O
Location symbol missing where required, or error in location symbol	L

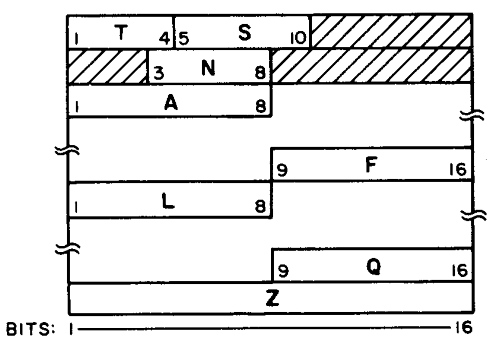
Error	Flag
Address of variable field expression not in sector being processed or sector zero (applicable only in load mode).	S
Relocation assignment error	R
Symbol table or literal table exceeded.	X
Major formatting error	F
Unclassified error in variable field of multiple field pseudo-operation (i. e., DEC, OCT, etc.)	V
Improper use of or error in index field	T
Undefined Symbol	U

Errors in a field generally result in that field being assembled as a 0. In the case of multiply defined symbols, the first symbol definition is used. If the operation code is illegal for computer configuration, the assembly is performed and the illegal codes are flagged with an "O."

OBJECT PROGRAM FORMAT

The object is used by DAP-16 when assembling programs in the desectorizing mode. This mode allows for relocatable main programs and subroutines in addition to absolute programs. Data are outputted in blocks composed of a parameter byte, followed by a data-word byte, then a logical difference checksum. There are eight block types (0-7) which are identified by bits 1 through 4 of the first word in the block. Block type zero is further subdivided into subblocks which are identified by bits 5 through 10 of the first word in the block. The following paragraphs contain a description of the various block types and their format.

Block Type 0-0 Subprogram Name



T = 0, the block type

S = 0, the subblock type

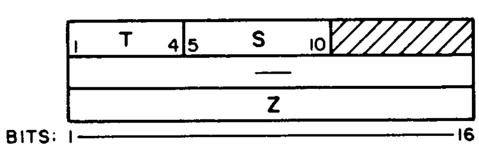
N is number of 16-bit words in the block including the checksum and control words

A-F is six-character name of the first entry point into the subprogram

L-Q is six-character name of the last entry point into the subprogram in this block.

Z is checksum for all words in block except for the checksum word

Block Types 0-1, 0-2, and 0-3 Special Action



T = 0

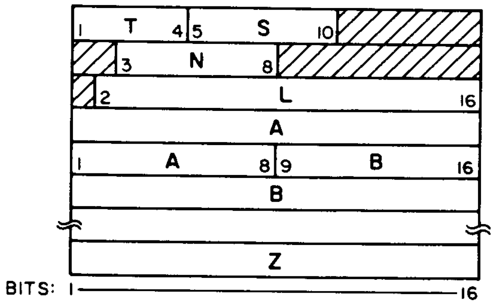
S = 1, turn off non-load flag

S = 2, turn on chain flag

S = 3, end-of-job

Z is checksum

Block Type 0-4 Data



T = 0

S = 4

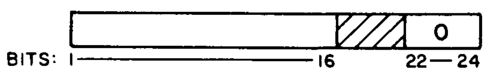
N is number of 16-bit words in the block

L is 15-bit address of location into which the first data is to be loaded. Successive words are loaded into location L + 1, L + 2, etc.

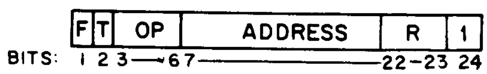
A, B. . . are data words in 24-bit format

Z is checksum

The data-word bytes have several formats, depending upon the last three bits of the byte. These formats are as follows.



Unmodified data generic or shift



Address is known and to be desectorized

R = 0, absolute

R = 1, positively relocatable

R = 3, negatively relocatable

Symbolic address, to be desectorized when the address is known

Bit 8 = 0, this is the last symbol number associated with the address

Bit 8 = 1, the following symbol number is also associated with the address. The following symbol number appears in bits 8-21 of the next data word providing the current word is not the last word in the current data block. If the current word is the last word in the current block, the symbol number appears in the next data block.

Address is known, do not desectorize

R = 0, absolute

R = 1, positively relocatable

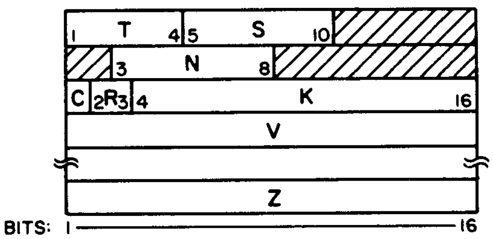
R = 3, negatively relocatable

Symbolic address, not to be desectorized when the address is known.

Bit 8 = 0, this is the last symbol number associated with the address.

Bit 8 = 1, the following symbol number is also associated with the address. The symbol number may appear in the next block if the current word is the last word in the current data block.

Block Type 0-10 Symbol Number Definition Block



T = 0

S = 10

C = 0, symbol is referred to only once

C = 1, symbol is referred to more than once

R = 0, absolute

R = 1, positively relocatable

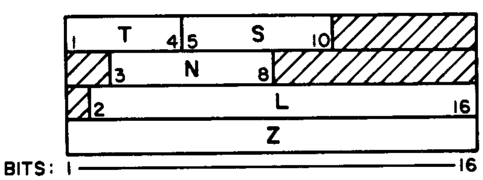
R = 3, negatively relocatable

K is 13-bit symbol number

V is 16-bit symbol value (positive or negative)

Z is checksum

Block Type 0-14 End



T = 0

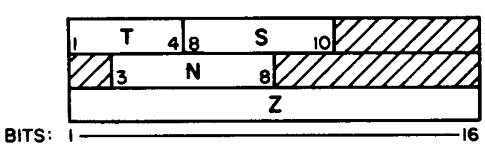
S = 14

N is number of 16-bit words in the block (always 4)

L is the jump address if this is the end of a main program. L is zero if this is the end of a subprogram

Z is checksum

Block Types 0-24, 30, 54, 60 Modes



T = 0

S = 24, relocatable mode

S = 30, absolute mode

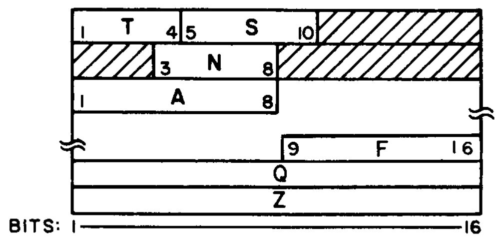
S = 54, enter extended-memory desectorizing mode

S = 60, leave extended-memory desectorizing mode

N is number of 16-bit words in the block (always 3)

Z is checksum

Block Type 0-44 Subprogram Call



T = 0

S = 44

N is number of 16-bit words in the block (always 7)

A-F is six-character name of the entry point

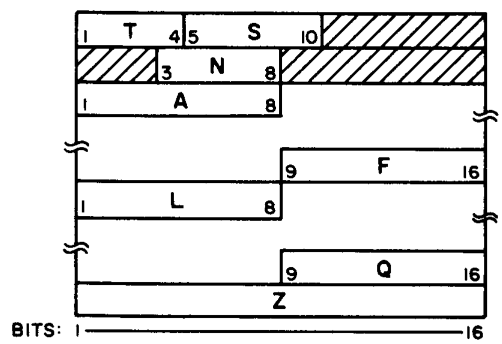
Q = 1, reference is not to be desectorized

Q = 0, reference is to be desectorized

Z is checksum

The last data word loaded is a reference to this subroutine name.

Block Type 0-50 Subprogram Entry Point Definition



T = 0

S = 50

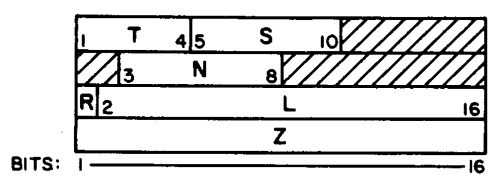
N is number of 16-bit words in the block

A-F is the first six-character name of this entry point into the subprogram

L-Q is the last six-character name of this entry point into the subprogram

Z is checksum

Block Type 0-64 Set Base Sector



T = 0

S = 64

N is number of 16-bit words in the block (always 4)

R = 0, absolute location

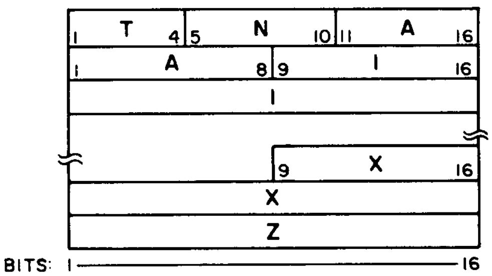
R = 1, relocatable location

L is 15-bit address of location at which the cross-sector indirect word table begins

Z is checksum

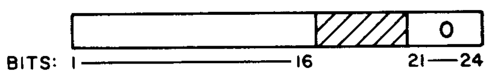
Block Types 0-20, 0-34, and 0-40 are illegal. They are reserved for internal functions of DAP-16.

Block Types 1 and 2 Program Words

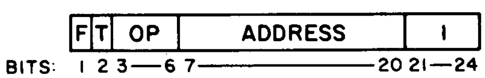


T = 1, absolute program words
T = 2, relative program words
N is number of 16-bit words in the block
I-X is 24-bit data words
Z is checksum

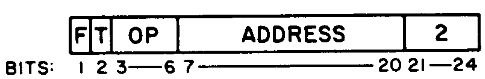
The data-word bytes in this block have several formats depending upon the last four bits of the byte. These formats are shown as follows.



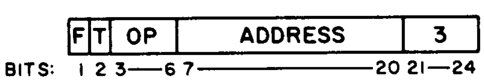
Load first 16 bits into memory unchanged



Address is not altered

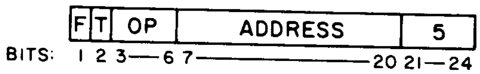


Address is positively relocated (add Δ)

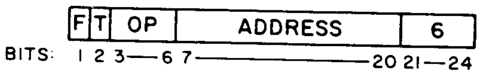


Address is negatively relocated (add Δ, complement)

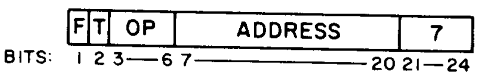
For types one through three, the address modified is interpreted as a 9-bit quantity. In case of an intersector reference, an indirect reference to sector zero is created.



Address is not altered



Address is positively relocated



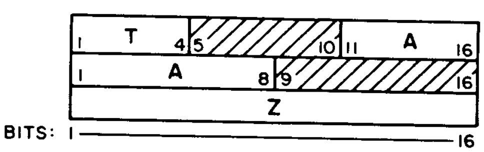
Address is negatively relocated

For types 5 through 7, the resultant address is merely combined with the F and T fields before loading.

U is all ONEs

V is relative address of an instruction in a string of instructions each of which uses the same symbol. The relative address (V) is supplied to each instruction requiring the address. The string may contain DAC pseudo-operations which accept a full 14-bit address and are distinguished from those instructions requiring a 9-bit address by their zero operation code. No instruction in a string may be desectorized into a base sector other than the currently active base sector.

Block Types 3 and 4 End Jumps



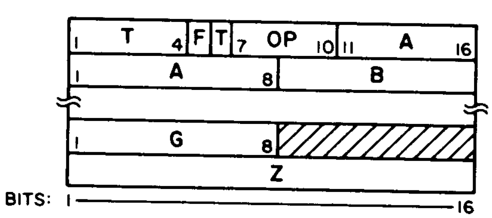
T = 3, jump address is absolute

T = 4, jump address is relative

A is jump address

Z is checksum

Block Types 5 and 7 Subroutine or Reference Call



T = 5, Subroutine call

T = 7, reference to an item in common

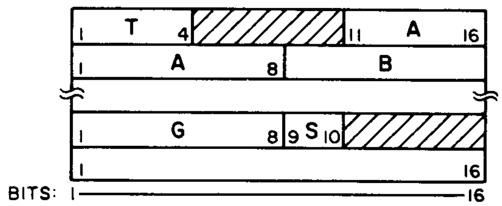
A is address of instruction. If T = 5 the address is relative to the common base sector

B-G is six-character name of subroutine or common item

Z is checksum

If C = 5, the operation code is JST*

Block Type 6 Subroutine or Common Block Definition



T = 6

A is entry print relative to the beginning of the subroutine if S = 0 or 2

= size of the common block if S = 1 or 3

B-G is six-character name of subroutine or common block

S = 0, subroutine definition

S = 1, common block definition

S = 2, subroutine definition

S = 3, data storage in common follows this block

Z is checksum

PROGRAMMING EXAMPLES

This example is not intended to be executable but is given to illustrate various DAP-16 pseudo-operation features.

```
                                  * C500-001-6504 (DAP-TEST)    CONTROL NUMBER 7011657     REV. A
                                  * START OBJECT PROGRAM AT OCTAL 10
                                  *
                                  * PROGRAM SHOULD TYPE 'O.K.' AND HALT
                                  *
                                  *
                                       ORG   '210
00210  0 02 00274                      LDA   00          COMPUTE CHECKSUM
00211  0 04 00000                      STA   0                  *
00212  140040                          CRA                      *
00213  1 05 00276                      ERA   TT+1,1             *
00214  0 12 00000                      IRS   0                  *
00215  0 01 00213                      JMP   *-2                *
00216  0 11 00277                      CAS   CKSM               *
00217  000000                          HLT                      *
00220  0 01 00222                      JMP   *+2                *
00221  000000                          HLT               WRONG SUM
00222  0 02 **          T              LDA   =-3         RIGHT SUM
00223  0 04 00000                      STA   0           TYPE 'O.K.'
00224  1 02 00245      TTTT            LDA   MSG+3,1            *
00225  07 0104                         SKS   '104               *
00226  0 01 00225                      JMP   *-1                *
00227  03 0104                         OCP   '104               *
00230  0406 70                         ARR   8                  *
00231  17 0004                         OTA   4                  *
00232  0 01 00231                      JMP   *-1                *
00233  0416 70                         ALR   8                  *
00234  17 0004                         OTA   4                  *
00235  0 01 00234                      JMP   *-1                *
00236  0 12 00000                      IRS   0                  *
00237  0 01 00224                      JMP   TTTT               *
00240  000000                          HLT               TEST COMPLETE
00241  177775                          FIN

00242  106612          MSG             OCT   106612             CARRIAGE RETURN + LINE FEED
00243  147656                          BCI   2,O.K.
00244  145656
                                  *
                                  * THE FOLLOWING CHECKS DAP OPERATION
00245  0 02 00000      XX              LDA   +1-1+2-2+3-3
       177534          YY              EQU   -XX+1
       000011          ZZ              EQU   3+3+3
       000145          M               EQU   XX-'100
00246  0 177356                        DAC   YY-ZZ-M
00247  -1 00 00250                     PZE*  XX+3,1
00250  120240                          BCI   2,
00251  120240
00252  000001                          OCT   1,3,144,-4,77777
00253  000003
00254  000144
00255  177774
00256  077777
00257  071143                          DEC   .99E+30
00260  173346
00261  007320                          DEC   .99E-30
00262  050576
00263  177372                          DEC   -.002B-2,15.001B+5,15.001BB+5
00264  036001
00265  036001
00266  001422
00267  036545                          DEC   1234.E-5,.15EE-2
00270  013333
00271  035742
00272  046722
00273  170651
00274  -1 177724       OO              DAC*  -TT+T-1,1
00275  -1 00 00250     TT              PZE*  XX+3,1
00276  0 000000                        DAC   *-*
00277  105314          CKSM            OCT   105314
00312                                  BES   10
00312                                  BSS   10
       000324          LIM             EQU   *
                                       END
```

SECTION VI
STANDARD INPUT/OUTPUT LIBRARY

Discussions in this section for various routines are valid for the Input/Output Library as used on Honeywell Series 16 computers (Honeywell 316/416/516).

ASR-33/35 TAPE READER, ASCII (IAA, IAI, I$GA)

I$AA reads ASCII paper tape using the ASR-33/35 paper tape reader or keyboard. If I$AA is not initialized by I$AI, it assumes that the input buffer is 40 words long and that there are three tab settings corresponding to character positions 6, 12, and 30 (DAP-16 source format).

Calling Sequence

```
                    CALL    I$AI
Initialization  -   DEC     (number of words in input buffer)
                    DEC     (number of tabs in following table, if any)
                    DEC     TAB (1)
                    DEC     TAB (2)
                     :       :
                    DEC     TAB (n)
                    (Normal return)
Read data       -   CALL    I$AA
                    DAC     (Data buffer address)
                    (End of message return)
                    (Normal return)
```

Method

Refer to the program listing, Honeywell/CCD Doc. No. 189001000, for details on the method used.

ASR-33/35 TAPE READER, BINARY (IAB, IABI, I$AI)

I$AB reads binary paper tape by using the ASR-33/35 paper tape reader. This routine is initialized by using the I$ABI entry. The address of the 60-word buffer into which the binary data is read appears in the variable field following the CALL pseudo-operation.

Calling Sequence

```
Initialization  -   CALL    I$AI  or  I$ABI
                    DAC     (Data)
```

(End of message return)
(Normal return)

Read data - CALL I$AB
DAC (Data)
(End of message return)
(Normal return)

Errors

Oversize input record. Computer halts. Check input tape for correct control characters between records. Press START pushbutton to take normal return.

Method

Refer to the program listing, Honeywell/CCD Doc No. 189002000, for details on the method used.

ASR-33/35 TELETYPE - LISTING AND HEADING ROUTINES (OLL, OHH)

These routines type out listings on the ASR-33/35 teletype. O$LL is called to type a line of data and O$HH is called to type out a heading. This routine backscans each buffer to edit trailing blanks. Refer to Honeywell/CCD Doc. No. 180774000 for details on methods used.

Storage Requirements

O$LL and O$HH require 147_{10}(223_8) locations.

Calling Sequence

Listing	CALL	O$LL
	DAC	(Data line address)
	(Normal return)	
Heading	CALL	O$HH
	DAC	(Heading address)
	(Normal return)	

ASR-33/35 TAPE PUNCH, ASCII (OAA, OAI, O$AS)

O$AA punches ASCII paper tape using the ASR-33/35 paper tape punch. This routine assumes, if not initialized by O$AL, that the data buffer is 40 words long and that there are three tab positions corresponding to character positions 6, 12, and 30 (DAP-16 source format). The O$AS entry is used to punch end of message.

Calling Sequence

```
Initialization  -  CALL    O$AI
                   DEC     (Number of words in data buffer)
                   DEC     (Number of tabs in following table, if any)
                   DEC     TAB (1)
                   DEC     TAB (2)
                    :       :
                   DEC     TAB (n)
                   (Normal return)
Data            -  CALL    O$AA
                   DAC     (Data buffer address)
                   (Normal return)
End of          -  CALL    O$AS
message
```

Method

Refer to program listing, Honeywell/CCD Doc. No. 189003000, for details on method used.

ASR-35 TAPE PUNCH, ASCII (OAA, OAI, O$AS)

Same as O$AA for ASR-33 except Honeywell/CCD Doc. No. is 180431000.

ASR-33 TAPE PUNCH, BINARY (OAB, OAS)

O$AB punches binary paper tape using the ASR-33 paper tape punch. The O$AS entry is used to punch end of message.

Calling Sequence

```
Punch data      -  CALL    O$AB
                   DAC     (Data)
                   (Normal return)
End of          -  CALL    O$AS
message
                   (Normal return)
```

Method

Refer to program listing, Honeywell/CCD Doc. No. 189004000, for details on method used.

ASR-35 TAPE PUNCH, BINARY (OAB, OAS)

Same as O$AB for ASR-33 except Honeywell/CCD Doc. No. is 180432000.

PAPER TAPE READER, ASCII (IPA, IPI)

I$PA reads paper tape in ASCII format by using the high-speed paper tape reader. The I$PI entry is used for initialization. If not initialized, the read routine assumes that the input buffer is 40 words long and that there are three tab settings corresponding to character positions 6, 12, and 30 (DAP-16 source format).

Calling Sequence

```
Initialization  -  CALL   I$PI
                   DEC    (Number of words in input buffer)
                   DEC    (Number of tabs in following table, if any)
                   DEC    TAB (1)
                   DEC    TAB (2)
                    :      :
                   DEC    TAB (n)
                   (Normal return)
Read data       -  CALL   I$PA  or  I$GA
                   DAC    (Data buffer address)
                   (End of message return)
                   (Normal return)
```

Errors

When input records exceed the buffer size the excess characters are lost.

Method

Refer to program listing, Honeywell/CCD Doc. No. 189006000, for details on method used.

PAPER TAPE READER, BINARY (IPB, IPBI)

I$PB reads paper tape in binary format by using the high-speed paper tape reader. The address of the 60-word buffer into which the binary information is read appears in the variable field following the CALL pseudo-operation.

Calling Sequence

```
CALL   I$PB   (or  I$PBI)
DAC    (Data) (Binary data address)
(End of message return)
(Normal return)
```

Errors

Oversize input record. Computer halts. Check input tape for correct control characters (X-OFF, RUBOUT, START OF MESSAGE) between records. Press START pushbutton to take normal return.

Method

Refer to the program listing, Honeywell/CCD Doc. No. 189007000, for details on the method used.

OUTPUT TO ASR PRINTER IN ASCII (O$AL)

To type out listings on the ASR.

Calling Sequence

```
CALL    O$AH    ASR heading routine
DAC     HEAD    Heading address
(Return)
```

In order to type a heading along with the page number at the top of each new page, a DAC must appear in the operation field followed by the address of a 23-word block in the variable field. This 23-word block serves as the heading.

The normal return is taken after the heading block has been filled, the page number set to one, and the line count set to zero.

```
CALL    O$AL    ASR listing routine
DAC     DATA    Line address
(Return)
```

In order to type a line of output a DAC must appear in the operation field followed by the address of a 60-word block in the variable field.

The normal return is taken when the line has been typed out.

Errors

No error checking is performed in this routine.

Method

This routine backscans each buffer, starting with the last word of the buffer, to edit trailing blanks. Refer to Honeywell/CCD Doc. No. 189005000 for details of method used.

OUTPUT ASCII INFORMATION TO UNBUFFERED SHUTTLE LINE PRINTER (OLA, OLHS, OLE, OLES, OLI, OLC, O$PN)

To print ASCII information on the Unbuffered Shuttle Line Printer.

Calling Sequence

CALL	O$LH	Initialization/heading routine
DAC	HEAD	Heading address
CALL	O$LE	Eject page
(Return)		
CALL	O$LA	Line printer routine
DAC	DATA	Line address
(Return)		
XAC	O$LI	Size of header
XAC	O$LC	Maximum lines per page
XAC	O$PN	Next page number

Method

Refer to Honeywell/CCD Doc. No. 180768000 for details on method used.

Other Routines Called

O$LB

I/O BUS TO SHUTTLE PRINTER CONFIGURATION ROUTINE (O$LB)

To configure O$LA for I/O Bus Operation.

Calling Sequence

LDA	ADDR	Starting address of 60 word buffer
CALL	O$LB	

Method

Refer to Honeywell/CCD Doc. No. 180769000 for details on method used.

PAPER TAPE PUNCH, ASCII (OPA, OGA, OPI, OPS, O$PLDR)

O$PA punches paper tape in ASCII format by using the high-speed paper tape punch. The package also has provisions for initialization (O$PI), punching end of record (O$PS), and punching leader (O$PLDR), depending on which entry is used. If not initialized, the punch routine assumes that the data buffer is 40 words long and that there are three tab settings corresponding to character positions 6, 12, and 30 (DAP-16 source format).

Calling Sequence

```
Data            -  CALL    O$PA  or  O$GA
                   DAC     (Data buffer address)
                   (normal return)
Initialization  -  CALL    O$PI
                   DEC     (Number of words in data buffer)
                   DEC     (Number of tabs in following table, if any)
                   DEC     TAB (1)
                   DEC     TAB (2)
                    :       :
                   DEC     TAB (n)
                   (Normal return)
End of record   -  CALL O$PL
                   (Normal return)
Leader          -  CALL O$PLDR
                   (Normal return)
```

Method

Refer to the program listing, Honeywell/CCD Doc. No. 189008000, for complete details on use.

PAPER TAPE PUNCH, BINARY (OPB, OPS, O$PLDR)

O$PB punches paper tape in binary format using the high-speed paper tape punch. This package has provisions for punching end of message by using the O$PS entry, and punches leader by the O$PLDR entry.

Calling Sequence

```
Data            -  CALL    O$PB
                   DAC     Data (Binary data address)
                   (Normal return)
End of message  -  CALL    O$PS
                   (Normal return)
Punch Leader    -  CALL    O$PLDR
                   (Normal return)
```

Method

Refer to the program listing, Honeywell/CCD Doc. No. 189009000, for complete details on method used.

PAPER TAPE PUNCH - LISTING AND HEADING ROUTINES (OPL, OPH)

O$PL punches listings on the paper tape punch. O$PL is called to punch a line of output and O$PH is called to punch a heading. This routine backscans each buffer to edit trailing blanks. Refer to Honeywell/CCD Doc. No. 181479000 for details of use.

Calling Sequence

```
Listing         -  CALL    O$PL
                   DAC     Data (Line address)
                   (Normal return)
```

Heading - CALL O$PH
DAC Head (Heading address)
(Normal return)

CARD READER, ASCII (ICA, IGA)

I$CA reads ASCII (Hollerith) cards using the Honeywell 316/516 card reader. One card is read on each I$CA entry. The data is stored two characters per word in a 40-word data buffer after being converted from the 6-bit code generated by the card reader to the 8-bit ASCII code.

Calling Sequence

```
CALL    I$CA or I$GA
DAC     (Data buffer address)
(End-of-file return)
(Normal return)
```

Errors

Card reader hopper empty, stacker full, jammed, validity, read check, or in manual. Upon detection of an error, the routine will automatically set up to re-read the card creating the error.

Method

Refer to the program listing, Honeywell/CCD Doc. No. 189011000, for details on method used.

CARD READER, BINARY (ICB, IGB)

I$CB (I$GB) reads column binary cards using the Honeywell 316/516 card reader.

Calling Sequence

```
CALL                 I$CB or I$GB
DAC                  (Data address - first word of 6-word block)
(End-of-file return)
(Normal return)
```

Errors

Card reader hopper empty, stacker full, jammed, read check, or in manual. Upon detection of an error, the routine will automatically set up to re-read the card creating the error.

Method

Refer to the program listing, Honeywell/CCD Doc. No. 180609000, for details on method used.

MAGNETIC TAPE READ PACKAGE (I$MA-U, I$MB, I$MC)

To read a magnetic tape in one of three modes depending on which entry is used. The entry mnemonics and corresponding read modes are:

I$MA	Read in BCD mode, 2 characters per word
I$MB	Read in binary mode, 2 characters per word
I$MC	Read in binary mode, 3 characters per word

Calling Sequence

```
CALL    I$Mx     (where x is A, B, or C)
DAC     BUFA     (Buffer address)
DEC     WC       (Word count, expressed as a decimal number)
DEC     N        (Logical type unit, expressed as a decimal number)
(Record unreadable return)
(End-of-tape return)
(End-of-file return)
(Normal return)
```

Method

Refer to the program listing, Honeywell/CCD Doc. No. 182604000, for details on the method used.

Other Routines Called

M$UNIT

MAGNETIC TAPE CONTROL PACKAGE (CMR, CFR, CBR, CFF, C$BF)

This routine performs one of five magnetic tape control functions depending on which entry is used. The entry mnemonics and corresponding control functions are:

C$MR	-	Rewind tape
C$FR	-	Forward space one record
C$BR	-	Backspace one record
C$FF	-	Forward space one file
C$BF	-	Backspace one file

Calling Sequence

For CMR, CFF, or C$BF -

```
CALL    C$xx     (where xx is MR, FF, or BF)
DEC     N        (Logical tape unit)
(Normal return)
```

For C$FR, or C$BR -

```
CALL    C$xx    (where xx is FR or BR)
DEC     N       (Logical tape unit)
(End-of-file return)
(Normal return)
```

Method

Refer to the program listing, Honeywell/CCD Doc. No. 182606000, for details on the method used.

Other Routines Called

M$UNIT

MAGNETIC TAPE WRITE PACKAGE (O$MA-U, O$MB, OMC, OME)

These routines are used to write a binary tape in one of three modes or to write an end-of-file depending on which entry is used. The entry mnemonics and corresponding write modes are:

O$MA	-	Write in BCD mode, 2 characters per word
O$MB	-	Write in binary mode, 2 characters per word
O$MC	-	Write in binary mode, 3 characters per word
O$ME	-	Write end-of-file

Calling Sequence

For writing a magnetic tape in a BCD or binary mode:

```
CALL    O$Mx    (Where x is A, B, or C)
DAC     BUFA    (Buffer address)
DEC     WC      (Word count, expressed in decimal)
(End-of-tape return)
(Normal return)
```

For writing an end-of-file on magnetic tape:

```
CALL    O$ME    Call subroutine
DEC     N       Logical tape unit, expressed in decimal
```

Method

Refer to the program listing, Honeywell/CCD Doc. No. 182605000, for details on the method used.

Other Routines Called

M$UNIT

MAGNETIC TAPE UNIT CONVERSION ROUTINE (M$UNIT-U)

M$UNIT-U provides a physical tape number associated with a logical tape when called by the magnetic tape read, write, and control routines. This routine requires manual configuration after loading. Refer to Operator's Guide for detail instructions for configuring logical to physical and I/O channel assignment.

Calling Sequence

To assign a physical tape number:

CALL	M$UNIT	Call the subroutine Return with physical number in A-register 14-16

To assign a channel number:

CALL	M$CHAN	Call the subroutine Return with channel number in A-register 14-16

To determine the channel type:

LDA	M$TY	Load A from external name
Where:	A = 0	I/O Bus
	A = 1	DMC
	A = 2	DMA

Method

Refer to the program listing, Honeywell/CCD Doc. No. 180228000, for details on the method used.

CONVERT IBM TAPE CODE TO ASCII (C$6T08)

C$6T08 converts standard magnetic tape code to ASCII. The data buffer is assumed to initially contain data in IBM tape code, stored two characters per word in bit positions 1-6 and 7-12 (data in bits 13-16 is ignored). After conversion, the contents of the buffer is replaced on a character-by-character basis. The character originally occupying bit positions 1-6 of a word occupies bit positions 1-8 of the same word. The character originally occupying bit positions 7-12 occupies bit positions 9-16.

Calling Sequence

```
CALL      C$6T08
DAC       (Buffer address)
DEC       (Number of words in buffer)
(Return)
```

Method

Conversion is made by table look-up. Refer to the program listing, Honeywell/CCD Doc. No. 180091000, for details on the method used.

CONVERT ASCII TO IBM TAPE CODE (C$8T06)

C$8T06 converts ASCII to standard magnetic tape code. The data buffer is assumed to initially contain ASCII data stored in bits 1-8 and 9-16. After conversion, the contents of the buffer is replaced on a character-by-character basis. The character originally occupying bit positions 1-8 of a word, after conversion, occupies bit positions 1-6 of the same word. The character originally occupying bit positions 9-16 occupies bit positions 7-12. Bit positions 13-16 of each word are set to zero.

Calling Sequence

```
CALL    C$8T06
DAC     (Buffer address)
DEC     (Number of words in buffer)
(Return)
```

Method

Conversion is made by table look-up. Refer to program listing, Honeywell/CCD Doc. No. 180082000, for details on program listing.

MOVING HEAD DISC FILE DRIVER (M$IO)

To support basic read/write capabilities for the Moving Head Disc File(available for the DDP-516 only).

USE

Loading

The moving head disc driver is a relocatable object program loadable by the standard DDP-416/516 linking loaders. It contains references to two external names: PAR1 and PAR2. The purpose of these two parameters is to configure the driver for DMC or DMA and the appropriate subchannel. The initialization entry of the driver takes the two parameters and makes the appropriate modifications. If the initialization entry is not called, the disc driver assumes a default condition of DMA subchannel 1.

It is the user's responsibility to configure his own disc. He may easily do this by creating a subroutine satisfying the two external references. This subroutine would then be loaded after the disc driver. The following is a sample:

```
        SUBR  PAR1
        SUBR  PAR2
        REL
PAR1    OCT   ARG1    If ARG1=0, DMA, otherwise DMC
PAR2    OCT   ARG2    Subchannel, DMA = 1-4, DMC = 1-16
        END
```

Calling Sequences

```
Call M$IN              Initialization call to configure disc

Call M$IO              Read/write call
OCT ARG1
OCT ARG2
DAC ARG3
OCT ARG4
ERROR RETURN
NORMAL RETURN
```

Arguments

ARG1 - Broken down as follows:

Bit 1	0 for read, 1 for write
Bits 2-3	Disc unit number (0-3) octal
Bits 4-7	Disc head number (0-11) octal
Bit 8	Disc control unit (0-310) octal
Bits 9-16	Cylinder (0-310) octal

ARG2 - 16 bit record address within the track

ARG3 - 15 bit buffer address for data (words one and two are utilized by the driver)

ARG4 - Data record size (if zero, 256 (decimal) is assumed)

Storage Capacity

The storage capacity of each option is as follows:

Unit	9433	9433A
Words Per Track	1,800	1,800
Words Per Surface	360,000	180,000
Words Per Device	3,600,00	1,800,00
Words Per Option	14,400,000	7,200,000

Method

Refer to program listing, Honeywell/CCD Doc. No. 180616000, for details on method used.

FIXED HEAD DISC FILE I/O DRIVER (D$IO)

To provide the user with an input/output driver for the Fixed Head Disc File that will configure the disc, format, and write the disc, and read and write records of varying length to and from the disc.

USE

Loading

The fixed head disc driver is a relocatable object program loadable by the standard DDP-416/516 linking loaders. It contains reference to two external names: PAR1 and PAR2. The purpose of these two parameters is to configure the driver for a DMA or DMC and the appropriate subchannel. The initialization entry uses the two parameters to configure the disc.

It is the user's responsibility to configure his own disc. He may easily do this by creating a subroutine which satisfies the two external names. This subroutine would be loaded after the disc driver. The following is a sample:

```
        SUBR   PAR1
        SUBR   PAR2
        REL
PAR1    OCT    ARG1          If ARG1=0, DMA, otherwise DMC
PAR2    OCT    ARG2          Subchannel, DMA1-4, DMC1-16
        END
```

NOTE

The user must allocate a buffer two words larger than the number of words which are to be transferred. The first two words of the buffer are used for the DMA setup words. The third through the nth words are transferred to the disc. The additional two words are required whether the channel used is a DMA or DMC.

Calling Sequences

CALL	C$FD	Format and write
OCT	ARG1	Bit 1: 0 - Read 1 - Write Bit 2: 0 - No checksum 1 - Checksum Bits 3-5: Not used Bits 6-16: Absolute track address
OCT	ARG2	Number of tracks to be formatted
DAC	ARG3	Buffer address
OCT	ARG4	Record length
ERROR RETURN		
NORMAL RETURN		
CALL	C$DI	Configure the fixed head disc file
CALL	D$IO	Read/write the fixed head disc file
OCT	ARG1	Bit 1: 0 - Read 1 - Write Bit 2: 0 - No checksum 1 - Checksum Bits 3-5: Not used Bits 6-16: Absolute track address
OCT	ARG2	Bits 1-8: Not used Bits 9-16: Record number
DAC	ARG3	Buffer address
OCT	ARG4	Record length
ERROR RETURN		
NORMAL RETURN		

Control Instructions

OCP	'322	Select DMA or DMC operation
OCP	'722	Select I/O bus operation
OCP	'422	Stop data transfer/acknowledge interrupt
SKS	'122	Skip if fixed head disc file is ready
SKS	'222	Skip if fixed head disc file has not detected data transfer error
SKS	'322	Skip if fixed head disc file has not detected an access error
SKS	'422	Skip if fixed head disc file is not interrupting
INA	'022	Input from fixed head disc file if ready
INA	'1022	Clear A-register and input from fixed disc file
OTA	'0022	Output data to the fixed head disc file

Set-Up Control Words

First Word

Bit 1:	0 for read, 1 for write
Bit 2:	Not used
Bit 3:	Device address 0 - Selects Disc one 1 - Selects Disc two
Bits 4-6:	Device address
Bits 7-12:	Track address (64/surface)
Bits 13-16:	Must be zero

Second Word

Bits 1-4:	Not used

Error Conditions

The D$IO routine executes an error return with the error condition noted in the A-register if its contents are the following:

A-register	Error condition
000001	Character error
000002	Checksum error
000004	Access error
000010	Data error, parity or timing
000020	Record number error
000040	Record length error

Record Format

Word 1	Record number	
Word 2	Bit 1:	Beginning of file mark
	Bits 9-16:	Record number
Word 3	Checksum (zero if none)	
Word 4	Bits 1-8:	End of record gap
	Bits 9-16:	Ones
Words 5-8	64 bits	
Words 9-N	Data	
Words N-1	End of record gap	

Method

Refer to program listing, Honeywell/CCD Doc. No. 180617000, for details on method used.

SECTION VII
MATHEMATICAL LIBRARY

All mathematical routines in the Honeywell 316/516 library are listed in Table 7-1. Each routine is listed alphabetically according to the function that it performs. Information given for each routine includes a mnemonic name, calling sequence, mode, errors, accuracy and timing (where available), storage locations required, and other routines used.

CALLS AND ARGUMENTS

The actual mnemonic name for a routine is given in the calling sequence in column 3. The routine identification in column 2 is not necessarily the entry for the routine indicated in column 1, but rather the identification of the routine that contains it.

After each call, in column 3, is the statement DAC Arg (1, 2, or n). DAC Arg 1 indicates that the program requires only one argument and the address of that argument appears to the right of the DAC. DAC Arg 2 indicates that the program requires two arguments. In this case, the first argument is in the appropriate accumulator and the address of the second argument appears to the right of the DAC. DAC Arg n indicates that the program requires more than two arguments. The first argument is in the appropriate accumulator, the address of the second is to the right of the first DAC, and the following lines contain additional DAC statements with the addresses of the additional arguments. There are four accumulators which are described below.

The single precision or real accumulator includes registers A and B, with the sign in A_1, the exponent in A_{2-9}, and the fraction in A_{10-16} and B_{1-16}. All accumulators are now relocatable.

The complex accumulator may be four relocatable memory locations AC1 to AC4. The sign of the real part is in bit position 1 of AC1, the exponent is in bit positions 2-9 of AC1, and the fraction is in bit positions 10-16 of AC1, and bit positions 1-16 of AC2. The imaginary part of the complex number occupies words AC3 and AC4 in the same manner.

NOTE

The integer accumulator is register A.

When FORTRAN IV is not being used, and an integer or single precision subroutine that requires more than one argument is required, place the first argument in register A, or registers A and B by means of LDA, or subroutine L$22.

For integer:

LDA	(address of integer variable)

For single precision:

CALL	L$22
DAC	ARG1

With double precision and complex subroutines, load the accumulators by means of the L$66 and L$55 subroutines.

For double precision:

CALL	L$66
DAC	(address of first word of double precision argument)

For complex:

CALL	L$55
DAC	(address of first word of complex argument)

Column 4, Mode, gives a symbolic representation of the mathematical function accomplished by each routine. Abbreviations that are used are defined as:

C	Complex number
R	Single precision number
I	Integer
D	Double precision number

The symbolic expressions given are interpreted in the conventional mathematical manner. The portion of the expression to the left of the equal sign is the result of the function and the portion on the right is the actual function performed. For example, in the first expression in the table, R = CABS(C) would be read R is a function of C, where R is the resulting single precision number, CABS (or complex absolute value) is the function performed, and (C) is the input argument (a complex number).

The last column in Table 7-1 gives other routines used by the routine listed in column 1. For routines coded in DAP format "Other Routines Used" includes only those called by the CALL pseudo-operation. For routines that are coded in FORTRAN, routines that are called by the FORTRAN compiler to fulfill the FORTRAN coding are given, in addition to those called by the FORTRAN source coding.

An explanation of conventions used throughout the library for transferring arguments to and from routines is presented in FORTRAN IV Library Introduction (Doc. No. 180092000). The FORTRAN IV Manual (Doc. No. 130071364) describes the distinction between functions and subroutines and provides instructions for writing programs that call both.

Table 7-1.
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Complex:					
Absolute value	CABS	Call CABS, DAC Arg 1	R=CABS(C)	None	FAT, L22, M$22, H$22, A$22, SQRT
Add	A$55	Call A$22, DAC Arg 2	C=C+C	None	FAT, H55, L$22, A$22, H$22, L$55
Add single precision argument	A$52	Call A$52, DAC Arg 2	C=C+R	None	FAT, H55, L$22, H$22, L$55
Conjugate	CONJG	Call CONJ, DAC Arg 1	C=CONJG(C)	None	FAT, L22, H$22, N$22, L$55
Convert imaginary part to real	AIMAG	Call AIMAG, DAC Arg 1	R=AIMAG(C)	None	L$55, L$22
Cosine	CCOS	Call CCOS, DAC Arg 1	C=CCOS(C)	None	FAT, L55, H$55, CSIN, A$55
Divide	D$55	Call D$55, DAC Arg 2	C=C/C	None	FAT, H55, L$22, M$22, H$22, A$22, D$22, S$22, L$55
Divide by single precision argument	D$52	Call D$52, DAC Arg 2	C=C/R	None	FAT, H55, L$22, D$22, H$22, L$55
Exponential, base e	CEXP	Call CEXP, DAC Arg 1	C=CEXP(C)	None	FAT, EXP, H22, COS M$22, SIN, L$55
Load	L$55	Call L$55, DAC Arg 1	C=C	None	ARG$
Logarithm, base e	CLOG	Call CLOG, DAC Arg 1	C=CLOG(C)	None	FAT, L22, M$22, H$22, A$22, ALOG ATAN2, L$55
Multiply	M$55	Call M$55, DAC Arg 2	C=C*C	None	FAT, H55, L$22, M$22, H$22, S$22, A$22, L$55
Multiply by single precision argument	M$52	Call M$52, DAC Arg 2	C=C*R	None	FAT, H55, L$22, M$22, H$22, L$55
Negate	N$55	Call N$55, DAC Arg 1	C=C	None	H$55, L$22, N$22, H$22, L$55

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Raise to integer power	E$51	Call E$51, DAC Arg 1	C=C**1	None	FAT, H55, LABS, L$55, M$55, D$55
Sine	CSIN	Call CSIN, DAC Arg 1	C=CSIN(C)	None	FAT, EXP, H22, L$22, D$22, A$22, SIN, M$22, L$55, S$22, COS
Square root	CSQRT	Call CSQRT, DAC Arg 1	C=CSQRT(C)	None	FAT, ABS, H22, CABS, A$22, M$22, SQRT, L$22, SIGN, D$22, L$55
Store (hold)	H$55	Call H$55, DAC Arg 1	C=C	None	ARG$
Subtract	S$55	Call S$55, DAC Arg 2	C=C-C	None	FAT, H55, L$22, S$22, H$22, L$55
Subtract single precision argument	S$52	Call S$52, DAC Arg 2	C=C-R	None	FAT, H55, L$22, S$22, H$22, L$55
<u>Double Precision:</u>					
Fixed-point:					
Addition	DADD	Call DADD, DAC Arg 2	D=D+D	Overflow; return to call plus 2	None
Arctangent	DATNX1	Call DATNX1, DAC Arg 1	D=DATNX1(D)	None	TWOS, DDIV, DMPY, DADD, RODD
*Arctangent	DATNX2	Call DATNX2, DAC Arg 1	D=DATNX2(D)	None	TWOS, DDIVH, DMPYH, DADD, RODD
Cosine	DCOSX1	Call DCOSX1, DAC Arg 1	D=DCOSX1(D)	None	DSINX1
*Cosine	DCOSX2	Call DCOSX2, DAC Arg 1	D=DCOSX2(D)	None	DSINX2
Divide	DMPY	Call DDIV, DAC Arg 2	D=D/D	Divisor ≤ dividend	MPY, TWOS, DIV
*Divide	DMPYH	Call DDIVH, DAC Arg 2	D=D/D	Divisor ≤ dividend	TWOS
Exponential, base e	DEXEX1	Call DEXED1, DAC Arg 1	D=DEXED1(D)	None	DADD, RODD, DMPY, DDIV, DSUB
*Exponential, base e	DEXEX2	Call DEXEX2, DAC Arg 1	D=DEXEX2(D)	None	DADD, RODD, DMPYH, DDIVH, DSUB

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Exponential, base 2	DEX2X1	Call DEX2X1, DAC Arg 1	D=DEX2X1(D)	Arg ≥ 0	RODD, DMPY, DADD, DDIV, DSUB
*Exponential, base 2	DEX2X2	Call DEX2X2, DAC Arg 1	D=DEX2X2(D)	Arg ≥ 0	RODD, DMPYH, DADD, DDIVH, DSUB
Logarithm, base e	DLGEX1	Call DLGEX1, DAC Arg 1	D=DLGEX1	$1/e >$ argument $\geq e$	DLG2X1, DMPY, DSUB, DADD
*Logarithm, base e	DLGEX2	Call DLGEX2, DAC Arg 1	D=DLGEX2	$1/e >$ argument $\geq e$	DLG2X2, DMPYH, DSUB, DADD
Logarithm, base 2	DLG2X1	Call DLG2X1, DAC Arg 1	D=DLG2X1	Argument $<1/2$	DADD, RODD, DSUB, DDIV, DMPY
*Logarithm, base 2	DLG2X2	Call DLG2X2, DAC Arg 1	D=DLG2X2	Argument $>1/2$	DADD, RODD, DSUB, DDIVH, DMPYH
Multiply	DMPY	Call DMPY, DAC Arg 2	D=D*D	None (overflow not possible)	MPY, TWOS, DIV
*Multiply	DMPYH	Call DMPYH, DAC Arg 2	D=D*D	None (overflow not possible)	MPY, TWOS
Round up binary number	RODD	Call RODD, DAC Arg 1	D=RODD(D)	None	None
Sine	DSINX1	Call DSINX1, DAC Arg 1	D=DSINX1(D)	None	DMPY, DADD
*Sine	DSINX2	Call DSINX2, DAC Arg 1	D=DSINX2(D)	None	DMPYH, DADD
Square Root	DSQRX1	Call DSQRX1, DAC Arg 1	D=DSQRX1(D)	None	TWOS, DMPY, DADD, DDIV, RODD
*Square root	DSQRX2	Call DSQRX2, DAC Arg 1	D=DSQRX2(D)	None	TWOS, DMPYH, DADD, DDIVH, RODD
Subtraction	DSUB	Call DSUB, DAC Arg 2	D=D-D	Overflow; return to call plus 2	None
Two's complement	TWOS	Call TWOS, DAC Arg 1	D=TWOS(D)	None	None

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Floating-point:					
Absolute value	DABS	Call DABS, DAC Arg 1	D=DABS(D)	None	FAT, L66, N$66
Add	A$66	Call A$66, DAC Arg 2	D=D+D	Overflow	ARG$, N$66, FER, H66, L$66
Add single precision argument	A$62	Call A$62, DAC Arg 2	D=D+R	None	EAT, H66, DBLE, A$66
Add integer to exponent	A$81	Call A$81, DAC Arg 2	D=A$81(1)	Overflow	F$ER
Arctangent, principal value	DATAN	Call DATAN, DAC Arg 1	D=DATAN(D)	None	F$AT, DABS, D$66, C$81, L$66, A$66, D$66, M$66, DSIGN
Arctangent, x/y	DATAN2	Call DATAN2, DAC Arg 2	D=DATAN2 (D, D)	Y=0	FAT, L66, H$66, F$ER, DATAN, S$66, A$66
Clear (zero, exponent)	Z$80	Call Z$80, DAC Arg 1	D=Z$80(D)	None	None
Convert exponent to integer	C$81	Call C$81, DAC Arg 1	I=C$81(D)	None	None
Convert to integer	C$61	Call C$61, DAC Arg 1	I=D	None	None
Convert to single precision (from pseudo accumulator)	C$62	Call C$62, DAC Arg 1	R=D	None	None
Cosine	DCOS	Call DCOS, DAC Arg 1	D=DCOS(D)	None	FAT, L66, A$66, H$66, DSIN
Divide	A$66	Call D$66, DAC Arg 2	D=D/D	Overflow	ARG$, N$66, FER, H66, L$66
Divide by single precision argument	D$62	Call D$62, DAC Arg 2	D=D/R	None	FAT, H66, DBLE, L$66, D$66
*Arithmetic add, subtract, mpy, + div.	A$66X	Call A$66X, DAC Arg 2	D=D*D	Overflow underflow division by zero	N$66, F$ER, H$66, L$66, N$66, ARG$
Exponential, base e	DEXP	Call DEXP, DAC Arg 1	D=DEXP(D)	None	FAT, M66, H$66, C$61, N$66, A$66, L$66, S$66, D$66, A$81

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Load	L$66	Call L$66, DAC Arg 1	D=D	None	ARG$
Logarithm, base e	DLOG	Call DLOG, DAC Arg 1	D=DLOG(D)	Negative or zero argument	F$AT, DLOG2, M$66, L$66, F$ER, C$81, C$16, H$66, S$66, Z$80, D$66, A$66
Logarithm, base 2	DLOG	Call DLOG2, DAC Arg 1	D=DLOG2(D)	Negative or zero argument	FAT, M66, L$66, F$ER, C$81, C$16, H$66, S$66, Z$80, D$66, A$66
Logarithm, base 10	DLOG10	Call DLOG10, DAC Arg 1	D=DLOG10(D)	Negative or zero argument	F$AT, DLOG2, M$66
Maximum value	DMAX1	Call DMAX1, DAC Arg n	D=DMAX1 (D1, D2, Dn)	None	L$66, H$66, S$66
Minimum value	DMIN1	Call DMIN1, DAC Arg n	D=DMIN1 (D1, D2, Dn)	None	L$66, H$66, S$66
Multiply	A$66	Call M$66, DAC Arg 2	D=D*D	Overflow	ARG$, N$66, H$66, L$66,
Multiply by single precision argument	M$62	Call M$62, DAC Arg 2	D=D*R	None	FAT, H66, DBLE, M$66
Negate	N$66	Call N$66, DAC Arg 1	D=D(-D)	None	None
Raise to double precision power	E$66	Call E$66, DAC Arg 2	D=D**D	None	FAT, H66, DLOG, M$66, DEXP
Raise to integer power	E$61	Call E$61, DAC Arg 2	D=D**I	None	FAT, H66, DABS, C$16, L$66, E$66, MOD, N$66
Raise to single precision power	E$62	Call E$62, DAC Arg 2	D=D**R	None	FAT, M62, H$66
Remainder	DMOD	Call DMOD, DAC Arg 2	D=DMOD(D, D)	None	FAT, L66, D$66, H$66, DINT, M$66, S$66, N$66
Sine	DSIN	Call DSIN, DAC Arg 1	D=DSIN(D)	None	FAT, L66, M$66, H$66, C$61, C$16, N$66, A$66, MOD, S$66

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Square root	DSQRT	Call DSQRT, DAC Arg 1	D=DSQRT(D)	None	FAT, L66, C$62, H$22, SQRT, C$26, H$66, D$66, A$66, A$81
Store (hold)	H$66	Call H$66, DAC Arg 1	D=D	None	ARG$
Subtract	A$66	Call S$66, DAC Arg 2	D=D-D	Overflow	ARG$, N$66, H$66, L$66
Subtract single precision argument	S$62	Call S$62, DAC Arg 2	D=D-R	None	FAT, H66, DBLE, S$66, N$66
Transfer sign of second argument to first	DSIGN	Call DSIGN, DAC Arg 2	D=DSIGN(D, D)	None	FAT, L66, N$66
Truncate fractional bits	DINT	Call DINT, DAC Arg 1	D=DINT(D)	None	L$66, N$66, A$66, S$66
Integer:					
Absolute value	IABS	Call IABS, DAC Arg 1	I=IABS(1)	None	None
Convert to double precision	C$16	Call C$16, DAC Arg 1	D=I	None	C$12, C$26
Convert (FORTRAN-generated) to single precision	FLOAT	Call FLOAT, DAC Arg 1	R=1	None	C$12
Convert to single precision	C$12	Call C$12, DAC Arg 1	R=1	None	A$22, N$22
Divide	D$11	Call D$11, DAC Arg 2	I=I/I	Division by zero -32768/-1	ARG$, F$ER
Maximum single precision value	MAX0	Call AMAX0, DAC Arg n	R=AMAX0 (11, 12, ..., 1n)	None	FLOAT
Maximum value	MAX0	Call MAX0, DAC Arg n	I=MAX0 (11, 12, ..., 1n)	None	FLOAT
Multiply	M$11	Call M$11, DAC Arg 2	I=I*I	Overflow Underflow	ARG$

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Positive difference	IDIM	Call IDIM, DAC Arg 2	I=I1-MIN (I1, I2)	None	None
Raise to integer power	E$11	Call E$11, DAC Arg 2	I=I**I	Consult listing	ARG$, M$11, F$ER
Remainder	MOD	Call MOD, DAC Arg 2	I=MOD(I, I)	None	D$11, M$11
Transfer sign of second argument to first	ISIGN	Call ISIGN, DAC Arg 2	I=ISIGN(I, I)	None	None
Single Precision:					
Fixed-Point:					
Arctangent	ATNX1	Call ATNX1, DAC Arg 1	R=ATNX(R)	None	MPY, ROND
*Arctangent	ATNX2	Call ATNX2, DAC Arg 1	R=ATNX2(R)	None	ROND
Cosine	COSX1	Call COSX1, DAC Arg 1	R=COSX(R)	None	SINX1
*Cosine	COSX2	Call COSX2, DAC Arg 1	R=COSX2(R)	None	SINX2
Divide	DIV	Call DIV, DAC Arg 2	R=R/R	Dividend ≥divisor	
Exponential, base e	EXEX1	Call EXEC1, DAC Arg 1	R=EXEC1(R)	None	MPY, DIV, ROND
*Exponential, base e	EXEX2	Call EXEX2, DAC Arg 1	R=EXEX2(R)	None	ROND
Exponential, base 2	EX2X1	Call EX2X1, DAC Arg 1	R=EX2X1(R)	Positive or zero argument	MPY, ROND, DIV
*Exponential, base 2	EX2X2	Call EX2X2, DAC Arg 1	R=EX2X2(R)	Positive or zero argument	ROND
Logarithm, base e	LGEX1	Call LGEX1, DAC Arg 1	R=LGEX1(R)	1/e> argument ≥e	LG2X1, MPY, ROND
*Logarithm, base e	LGEX2	Call LGEX2, DAC Arg 1	R=LGEX2(R)	1/e> argu- ≥e	LG2X2, ROND
Logarithm, base 2	LG2X1	Call LG2X1, DAC Arg 1	R=LG2X1(R)	5> argument ≥1	DIV, MPY
*Logarithm, base 2	LG2X2	Call LG2X2, DAC Arg 1	R=LG2X2(R)	5> argument ≥1	None

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Multiply	MPY	Call MPY, DAC Arg 2	R=R*R	None (overflow not possible)	MPYS
Round up binary number	ROND	Call ROND, DAC Arg 1	R=ROND(R)	None	None
Sine	SINX1	Call SINX1, DAC Arg 1	R=SINX1(R)	None	MPY
*Sine	SINX2	Call SINX2, DAC Arg 1	R=SINX2(R)	None	None
Square root	SQRX1	Call SQRX1, DAC Arg 1	R=SQRX1(R)	None	MPY, DIV
*Square root	SQRX2	Call SQRX2, DAC Arg 1	R=SQRX2(R)	None	None
Floating-Point:					
Absolute value	ABS	Call ABS, DAC Arg 1	R=ABS(R)	None	L$22, N$22
Add	A$22	Call A$22, DAC Arg 2	R=R+R	Overflow	ARG$, N$22, F$ER
Arctangent, principal value	ATAN	Call ATAN, DAC Arg 1	R=ATAN(R)	None	ARG$, D$22, N$22, M$22, A$22, S$22
Arctangent, y/x	ATAN	Call ATAN2, DAC Arg 2	R=ATAN2 (R, R)	None	ARG$, D$22, N$22, M$22, A$22, S$22
Convert (FORTRAN-generated) to double precision	DBLE	Call DBLE, DAC Arg 1	D=R	None	ARG$, D$22, N$22, M$22, A$22, S$22, C$26
Convert to integer or truncate fractional bits and convert to integer	IFIX	Call IFIX, INT or IDINT, DAC Arg 1	I=IFIX(R) I=R or I= IDINT(D)	None	L$22, C$21
Convert pair to complex	CMPLX	Call CMPLX, DAC Arg 2	C=R	None	None
Convert to complex format	C$25	Call C$25, DAC Arg 1	C=R	None	CMPLX
Convert to double precision	C$26	Call C$26, DAC Arg 1	D=R	None	None
Convert to integer	C$21	Call C$21, DAC Arg 1	I=R	Integer >15	N$22, A$22, F$ER

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Divide	D$22	Call D$22, DAC Arg 2	R=R/R	Overflow zero divisor	N$22, ARG$, RMPY, F$ER, DIV
Exponential, base e	EXP	Call EXP, DAC Arg 1	R=EXP(R)	Unnormalized argument, exponent overflow/underflow	ARG$, N$22, M$22, S$22, A$22, D$22, F$ER
Hyperbolic tangent	TANH	Call TANH, DAC Arg 1	R=TANH(R)	None	L$22, EXP, A$22, H$22, D$22
Load	L$22	Call L$22, DAC Arg 1	R=R	None	ARG$
Logarithm, base e	ALOG	Call ALOG, DAC Arg 1	R=ALOG(R)	Argument ≤0	ARG$, C$12, A$22, MULT, M$22, S$22, F$ER
Logarithm, base 10	ALOG	Call ALOG10, DAC Arg 1	R=ALOG10(R)	Argument ≤0	ARG$, C$12, A$22, MULT, M$22, S$22, F$ER
Maximum integer value	MAX1	Call MAX1, DAC Arg n	I=MAX1 (R1, R2, . . . , Rn)	None	L$22, H$22, S$22, IFIX
Maximum value	MAX1	Call AMAX1, DAC Arg n	R=AMAX1 (R1, R2, . . . , Rn)	None	L$22, H$22, S$22, IFIX
Minimum integer value	MIN1	Call MIN1, DAC Arg n	I=MIN1 (R1, R2, . . . , Rn)	None	L$22, H$22, S$22, IFIX
Minimum value	MIN1	Call AMIN1, DAC Arg n	R=AMIN1 (R1, R2, . . . , Rn)	None	L$22, H$22, S$22, IFIX
Multiply	M$22	Call M$22, DAC Arg 2	R=R*R	Overflow	N$22, ARG$, RMPY, F$ER, DIV
Positive difference	DIM	Call DIM, DAC Arg 2	R=DIM(R, R)	None	L$22, S$22
Raise to double precision power	E$26	Call E$26, DAC Arg 2	D=R**D	None	FAT, C26, H$66, DLOG, M$66, DEXP
Raise to integer power	E$21	Call E$21, DAC Arg 2	R=R**I	None	ARG$, M$22, D$22

*Operates with High Speed Arithmetic Unit option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Other Routines Used
Raise to single precision power	E$22	Call E$22, DAC Arg 2	R=R**R	None	ARG$, ALOG, M$22, EXP
Remainder	AMOD	Call AMOD, DAC Arg 2	R=AMOD(R, R)	None	L$22, D$22, AINT, M$22, N$22, A$22
Sine, Cosine	SIN, COS	Call SIN (or COS) DAC Arg 1	R=SIN(R) R=COS(R)	None	ARG$, N$22, M$22, S$22, A$22
Square root	SQRT	Call SQRT, DAC Arg 1	R=SQRT(R)	Negative argument	ARG$, DIV$, D$22, A$22, F$ER
Store (hold)	H$22	Call H$22, DAC Arg 1	R=R	None	ARG$
Subtract	A$22	Call A$22, DAC Arg 1	R=R-R	Overflow	ARG$, N$22, F$ER
Transfer sign of second argument to first	SIGN	Call SIGN, DAC Arg 2	R=SIGN(R, R)	None	L$22, N$22
Truncate fractional bits	AINT	Call AINT, DAC Arg 1	R=AINT(R)	None	L$22, N$22, A$22, S$22
Two's complement	N$22	Call N$22, DAC Arg 1	R=N$22(R)	None	None
Real multiplication and real division	M$22X	Call M$22X, DAC Arg 2	R=R*R	Overflow division by zero unnormalized divisor	N$22, ARG$, F$ER

*Operates with High Speed Arithmetic Unit option only.

APPENDIX A
NUMBERING SYSTEM AND TWO'S COMPLEMENT ARITHMETIC

Binary Numbering System

Sixteen-bit data words are stored in two's complement notation. The MSB of a data word may be considered to be the arithmetic sign of the number represented. The MSB is zero for positive (+) numbers and a one for negative (-) numbers. Bits 2 through 16 of the data word represent the value in binary form. Positive values thus range from zero (which always has a positive sign) to 32,767 as follows:

0	000 000 000 000 000	Zero
0	000 000 000 000 001	+1
0	000 000 000 000 010	+2
	'	'
	'	'
	'	'
	'	'
0	111 111 111 111 111	+32,767

Negative numbers are represented in two's complement form and always have a one in the sign bit position.

Two's Complement Arithmetic

The two's complement of a binary number is obtained by complementing (reversing) each bit and adding one. For example, the two's complement of +1, which represents -1, is obtained as follows:

+1	0	000 000 000	000 001
Complement	1	111 111 111	111 110
Add 1	0	000 000 000	000 001
Two's Complement (-1)	1	111 111 111	111 111

The number range for negative values is from -1 to -32,768 as follows:

1	111 111 111 111	111	(-1)
1	111 111 111 111	110	(-2)
1	111 111 111 111	101	(-3)
1	000 000 000 000	000	(-32,768)

If +1 is added to -1, the result is zero. Thus:

1	111 111 111 111	111	-1
0	000 000 000 000	001	+1
0	000 000 000 000	000	Zero

Note that a carry bit from the most significant position has been ignored. In two's complement arithmetic, if numbers of unlike signs are added together, carries from the MSB are disregarded.

Overflow

Overflow is the condition that occurs when two numbers of like signs are added together to produce a sum of a different sign. For example, adding +1 to ±32,767 produces a result larger than the capacity of a single data word.

0	111 111 111 111	111	(+32,767)
0	000 000 000 000	001	(+1)
1	000 000 000 000	000	

The different sign of the result defines an overflow condition.

Addition on the computer is performed by adding a quantity in the memory to a quantity in the A-register. True signed arithmetic takes place. Overflow conditions automatically result in the setting of the C-bit indicator, even though no carry is propagated from the sign position. In the preceding example, the C-bit indicator is set.

APPENDIX B
HONEYWELL 316/516 PERIPHERAL DEVICE CODES

Char.	ASCII Code*	Card Code Hollerith	Card Code Octal	MagTape Code
0	260	0	12[1]	12[1]
1	261	1	01	01
2	262	2	02	02
3	263	3	03	03
4	264	4	04	04
5	265	5	05	05
6	266	6	06	06
7	267	7	07	07
8	270	8	10	10
9	271	9	11	11
A	301	12-1	61	61
B	302	12-2	62	62
C	303	12-3	63	63
D	304	12-4	64	64
E	305	12-5	65	65
F	306	12-6	66	66
G	307	12-7	67	67
H	310	12-8	70	70
I	311	12-9	71	71
J	312	11-1	41	41
K	313	11-2	42	42
L	314	11-3	43	43
M	315	11-4	44	44
N	316	11-5	45	45
O	317	11-6	46	46
P	320	11-7	47	47
Q	321	11-8	50	50
R	322	11-9	51	51
S	323	0-2	22	22
T	324	0-3	23	23
U	325	0-4	24	24
V	326	0-5	25	25

Char.	ASCII Code*	Card Code Hollerith	Card Code Octal	MagTape Code
W	327	0-6	26	26
X	330	0-7	27	27
Y	331	0-8	30	30
Z	332	0-9	31	31
Space	240	Blank	20	20
!	241[2]	8-6	16	16
"	242[2]	0-8-7	37	37
#	243[2]	0-8-2	32	
$	244[2]	11-8-3	53	53
%	245[2]	12-8-5	75	75
&	246[2]	0-8-5	35	
'	247[2]	8-4	14	14
(	250[2]	0-8-4	34	34
)	251[2]	12-8-4	74	74
*	252[2]	11-8-4	54	54
+	253[2]	12	60	60
,	254	0-8-3	33	33
_	255	11	40	40
.	256	12-8-3	73	73
/	257	01	21	21
:	272	8-5	15	15
;	273	11-0	52	52
<	274[2]	11-8-7	57	57
=	275[2]	8-3	13	13
>	276[2]	8-7	17	17
?	277[2]	11-8-6**	56	
@	300[2]	Error	00	
[	333[3]	11-8-5	55	55
\	334[4]	12-8-6	76	76
]	335[5]	0-8-6	36	36
↑	336[6]	12-0	72	72
←	337	12-8-7	77	77

*Used by ASR and Line Printer

**End of File for Burroughs Card Reader

NOTES:

1. When writing magnetic tapes in even parity (BCD) mode, 00_8 is written as 12_8; conversely, when reading in even parity, 12_8 is read as 00_8.
2. Upper case characters on ASR-33/35
3. Upper case VT on ASR-33/35
4. Upper case FORM on ASR-33/35
5. Upper case M on ASR-33/35
6. Upper case N on ASR-33/35

APPENDIX C
SUMMARY OF STANDARD INSTRUCTIONS
(Listed in Alphabetical Order)

Mnemonic	Octal Code	Instruction	Type	Execution Time (μs) DDP-516	H316	Page
ACA	141216	Add C to A	G	0.96	1.6	2-4
ADD	06	Add	MR	1.92	3.2	2-4
ALR	0416	Logical Left Rotate	SH	0.96 + 0.48n	1.6 + 0.8n	2-5
ALS	0415	Arithmetic Left Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-5
ANA	03	AND to A	MR	1.92	3.2	2-4
AOA	141206	Add One to A	G	0.96	1.6	2-4
ARR	0406	Logical Right Rotate	SH	0.96 + 0.48n	1.6 + 0.8n	2-5
ARS	0405	Arithmetic Right Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-5
CAL	141050	Clear A, Left Half	G	0.96	1.6	2-10
CAR	141044	Clear A, Right Half	G	0.96	1.6	2-10
CAS	11	Compare	MR	2.88	4.8	2-11
CHS	140024	Complement A Sign	G	0.96	1.6	2-4
CMA	140401	Complement A	G	0.96	1.6	2-4
CRA	140040	Clear A	G	0.96	1.6	2-3
CSA	140320	Copy Sign and Set Sign Plus	G	0.96	1.6	2-4
ENB	000401	Enable Program Interrupt	G	0.96	1.6	2-11
ERA	05	Exclusive OR to A	MR	1.92	3.2	2-4
HLT	000000	Halt	G			2-11
IAB	000201	Interchange A and B	G	0.96	1.6	2-3
ICA	141340	Interchange Characters in A	G	0.96	1.6	2-10
ICL	141140	Interchange and Clear Left Half of A	G	0.96	1.6	2-10
ICR	141240	Interchange and Clear Right Half of A	G	0.96	1.6	2-10
IMA	13	Interchange Memory and A	MR	2.88	4.8	2-3
INA	54	Input to A	IO	1.92	3.2	2-14
INH	001001	Inhibit Program Interrupt	G	0.96	1.6	2-11
INK	000043	Input Keys	G	0.96	1.6	2-3
IRS	12	Increment, Replace and Skip	MR	2.88	4.8	2-12
JMP	01	Unconditional Jump	MR	0.96	1.6	2-12
JST	10	Jump and Store Location	MR	2.88	4.8	2-12
LDA	02	Load A	MR	1.92	3.2	2-3
LDX	15	Load X	MR	2.88	4.8	2-3
LGL	0414	Logical Left Shift	SH	0.96 + 0.48n	1.6 + 0.8	2-6

APPENDIX C (Cont)
SUMMARY OF STANDARD INSTRUCTIONS
(Listed in Alphabetical Order)

Mnemonic	Octal Code	Instruction	Type	Execution Time (μs) DDP-516	H316	Page
LGR	0404	Logical Right Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-6
LLL	0410	Long Left Logical Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-6
LLR	0412	Long Left Rotate	SH	0.96 + 0.48n	1.6 + 0.8n	2-7
LLS	0411	Long Arithmetic Left Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-7
LRL	0400	Long Right Logical Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-8
LRR	0402	Long Right Rotate	SH	0.96 + 0.48n	1.6 + 0.8n	2-8
LRS	0401	Long Arithmetic Right Shift	SH	0.96 + 0.48n	1.6 + 0.8n	2-9
NOP	101000	No Operation	G	0.96	1.6	2-12
OCP	14	Output Control Pulse	IO	1.92	3.2	2-14
OTA	74	Output From A	IO	1.92	3.2	2-14
OTK	171020	Output Keys	IO	1.92	1.6	2-3
RCB	140200	Reset C Bit	G	0.96	1.6	2-12
SCB	140600	Set C Bit	G	0.96	1.6	2-12
SKP	100000	Unconditional Skip	G	0.96	1.6	2-12
SKS	34	Skip if Ready Line Set	IO	1.92	3.2	2-15
SLN	101100	Skip if (A_{16}) is one	G	0.96	1.6	2-12
SLZ	100100	Skip if (A_{16}) is zero	G	0.96	1.6	2-12
SMI	101400	Skip if A Minus	G	0.96	1.6	2-12
SMK	74	Set Mask	IO	1.92	3.2	2-15
SNZ	101040	Skip if A Not zero	G	0.96	1.6	2-12
SPL	100400	Skip if A Plus	G	0.96	1.6	2-12
SRC	100001	Skip if C Reset	G	0.96	1.6	2-13
SR1	100020	Skip if Sense Switch 1 is Reset	G	0.96	1.6	2-12
SR2	100010	Skip if Sense Switch 2 is Reset	G	0.96	1.6	2-12
SR3	100004	Skip if Sense Switch 3 is Reset	G	0.96	1.6	2-12
SR4	100002	Skip if Sense Switch 4 is Reset	G	0.96	1.6	2-13
SSC	101001	Skip if C Set	G	0.96	1.6	2-13
SSM	140500	Set Sign Minus	G	0.96	1.6	2-4
SSP	140100	Set Sign Plus	G	0.96	1.6	2-4
SSR	100036	Skip if no Sense Switch Set	G	0.96	1.6	2-13
SSS	101036	Skip if any Sense Switch is Set	G	0.96	1.6	2-13
SS1	101020	Skip if Sense Switch 1 is Set	G	0.96	1.6	2-13
SS2	101010	Skip if Sense Switch 2 is Set	G	0.96	1.6	2-13
SS3	101004	Skip if Sense Switch 3 is Set	G	0.96	1.6	2-13
SS4	101002	Skip if Sense Switch 4 is Set	G	0.96	1.6	2-13
STA	04	Store A	MR	1.92	3.2	2-3
STX	15	Store X	MR	1.92	3.2	2-3

APPENDIX C (Cont)
SUMMARY OF STANDARD INSTRUCTIONS
(Listed in Alphabetical Order)

Mnemonic	Octal Code	Instruction	Type	Execution Time (μs) DDP-516	H316	Page
SUB	07	Subtract	MR	1.92	3.2	2-4
SZE	100040	Skip if a zero	G	0.96	1.6	2-13
TCA	140407	Two's Complement A	G	1.44	2.4	2-4

APPENDIX D
MAIN FRAME OPTION COMMANDS

Mnemonic	Octal Code	Instruction	Type	Execution Time	Page
Extended Addressing for 24K and 32K Memories - Model 516-05, 06					
DXA	000011	Disable Extended Mode	G	0.96	2-20
EXA	000013	Enable Extended Mode	G	0.96	2-20
Memory Parity - Model 516-07					
RMP	000021	Reset Memory Parity Error	G	0.96	2-21
SMK '0020	170020	Set Interrupt Mask (A_{15})	IO	1.92	2-21
SPN	100200	Skip if No Memory Parity Error	G	0.96	2-21
SPS	101200	Skip if Memory Parity Error	G	0.96	2-21
Memory Lockout - Model 516-08					
ERM	001401	Enter Restricted Mode	G	0.96	2-26
SMK '1320	171320	Set Relocation Register	IO	1.92	2-26
SMK '1420	171420	Set Lockout Mask 1	IO	1.92	2-26
SMK '1520	171520	Set Lockout Mask 2	IO	1.92	2-26
SMK '1620	171620	Set Lockout Mask 3	IO	1.92	2-26
SMK '1720	171720	Set Lockout Mask 4	IO	1.92	2-26
Direct Memory Access (DMA) - Model 516-21					
INA '1124	171124	Read Range Counter Channel 1	IO	1.92	2-34
INA '1224	171224	Read Range Counter Channel 2	IO	1.92	2-34
INA '1324	171324	Read Range Counter Channel 3	IO	1.92	2-34
INA '1424	171424	Read Range Counter Channel 4	IO	1.92	2-34
SMK '0124	170124	Load Address Counter Channel 1	IO	1.92	2-34
SMK '0224	170224	Load Address Counter Channel 2	IO	1.92	2-34
SMK '0324	170324	Load Address Counter Channel 3	IO	1.92	2-34
SMK '0424	170424	Load Address Counter Channel 4	IO	1.92	2-34
SMK '1124	171124	Load Range Counter Channel 1	IO	1.92	2-34
SMK '1224	171224	Load Range Counter Channel 2	IO	1.92	2-34
SMK '1324	171324	Load Range Counter Channel 3	IO	1.92	2-34
SMK '1424	171424	Load Range Counter Channel 4	IO	1.92	2-34

Mnemonic	Octal Code	Instruction	Type	Execution Time μs DDP-516	Execution Time μs H316	Page
Priority Interrupt · Model 316/516-25						
SMK '0020	170020	Set Standard Interrupt Mask	IO	1.92	3.2	2-36
SMK '0120	170120	Set Interrupt Mask Lines 1-16	IO	1.92	3.2	2-36
SMK '0220	170220	Set Interrupt Mask Lines 17-32	IO	1.92	3.2	2-36
SMK '0320	170320	Set Interrupt Mask Lines 33-48	IO	1.92	3.2	2-36
High-Speed Arithmetic Unit - Model 316/516-11						
DAD	06	Double Precision Add	MR	2.88	4.8	2-29
DBL	000007	Enter Double Precision Mode	G	0.96	1.6	2-29
DIV	17	Divide	MR	10.56 (max)	16 or 16.8 or 17.6 (max)	2-29
DLD	02	Double Precision Load	MR	2.88	4.8	2-29
DSB	07	Double Precision Subtract	MR	2.88	4.8	2-29
DST	04	Double Precision Store	MR	2.88	4.8	2-29
MPY	16	Multiply	MR	5.28	8.8	2-29
NRM	000101	Normalize	G	0.96+0.48n	1.6+0.8n	2-29
SCA	000041	Shift Count to A	G	0.96	1.6	2-29
SGL	00005	Enter Single Precision Mode	G	0.96	1.6	2-29
Real-Time Clock - Model 316/516-12						
OCP '0220	030220	Reset Interrupt Request and Stop Clock	IO	1.92	3.2	2-30
OCP '0020	030020	Reset Interrupt Request and Run Clock	IO	1.92	3.2	2-30
SKS '0020	070020	Skip if RTC not interrupting	IO	1.92	3.2	2-30
SMK '0020	070020	Set Interrupt Mask (A_{16})	IO	1.92	3.2	2-36

APPENDIX E
PERIPHERAL DEVICE COMMANDS

ASR33/35 Model 316/516-53/55

OCP	0004	Enable ASR-33/35 In Input Mode
OCP	0104	Enable ASR-33/35 In Output Mode
SKS	0004	Skip if ASR-33/35 is Ready
SKS	0104	Skip if ASR-33/35 is Not Busy
SKS	0404	Skip if ASR-33/35 is Not Interrupting
SKS	0504	Skip if Stop Code Was Not Read on ASR-33/35
INA	0004	Input in ASCII from ASR-33/35
INA	0204	Input in Binary from ASR-33/35
INA	1004	Clear Register A and Input in ASCII from ASR-33/35
INA	1204	Clear Register A and Input in Binary from ASR-33/35
OTA	0004	Output in ASCII to ASR-33/35
OTA	0204	Output in Binary to ASR-33/35
SMK	0020	Set Interrupt Mask (A_{11})

High Speed Paper Tape Reader - Model 316/516-50

OCP	0001	Start Paper Tape Reader
OCP	0101	Stop Paper Tape Reader
SKS	0001	Skip if Paper Tape Reader is Ready
SKS	0401	Skip if Paper Tape Reader is Not Interrupting
INA	0001	Input from Paper Tape Reader
INA	1001	Clear Register A and Input From Paper Tape Reader
SMK	0020	Set Interrupt Mask (A_9)

High Speed Paper Tape Punch - Model 316/516-52

OCP	0002	Enable Paper Tape Punch
OCP	0102	Turn Paper Tape Punch Power Off
SKS	0002	Skip if Paper Tape Punch is Ready
SKS	0102	Skip if Paper Tape Punch is Enabled
SKS	0402	Skip if Paper Tape Punch is Not Interrupting
OTA	0002	Output To Paper Tape Punch
SMK	0020	Set Interrupt Mask (A_{10})

Parallel Channels - Model 316/516-32, 33-34

INA	'0030*	Input to A-register
INA	'1030	Clear A-register and input to A-register
OTA	'0030	Output from A-register
OCP	'0030	Enable input mode (516-32, 34)
OCP	'0030	Enable output mode (516-33)

OCP	'0130	Enable output mode (516-34)
OCP	'0130	Device OCP 1 (516-32-33)
OCP	'0230	Device OCP 2 (516-32, 33, 34)
OCP	'0330	Device OCP 3 (516-32, 33, 34)
OCP	'0430	Device OCP 4 (516-33, 34)
OCP	'0530	Device OCP 5 (516-34)
OCP	'0630	Device OCP 6 (516-34 No DMC/DMA)
OCP	'0630	Enable DMC/DMA mode (516-32, 33, 34)
OCP	'0730	Reset DMC/DMA mode (516-32, 33, 34)
OCP	'1630	Enable DMC/DMA Auto-Switch Mode (516-32, 33, 34)
SKS	'0030	Skip if channel ready
SKS	'0130	Device SKS 1
SKS	'0230	Skip if first channel not reached end-of-range
SKS	'0330	Skip if not in auto-switch mode
SKS	'0430	Skip if no interrupt request
SKS	'0530	Device SKS 2
SKS	'0630	Device SKS 3
SMK	'0020	

*30 is the address of the first channel. Table below shows addresses for two other channels.

	Address	Mask Bit
First Channel	30_8	5
Second Channel	31_8	6
Third Channel	32_8	7

OCP/SKS - Model 316/516-29

OCP	'0034	Device OCP 00
OCP	'0134	Device OCP 01
OCP	'0234	Device OCP 02
OCP	'0334	Device OCP 03
OCP	'0434	Device OCP 04
OCP	'0534	Device OCP 05
OCP	'0634	Device OCP 06
OCP	'0734	Device OCP 07
OCP	'1034	Device OCP 10
OCP	'1134	Device OCP 11
OCP	'1234	Device OCP 12
OCP	'1334	Device OCP 13
OCP	'1434	Device OCP 14
OCP	'1534	Device OCP 15

OCP	'1634	Device OCP 16
OCP	'1734	Device OCP 17
SKS	'0034	Device SKS 00
SKS	'0134	Device SKS 01
SKS	'0234	Device SKS 02
SKS	'0334	Device SKS 03
SKS	'0434	Device SKS 04
SKS	'0534	Device SKS 05
SKS	'0634	Device SKS 06
SKS	'0734	Device SKS 07
SKS	'1034	Device SKS 10
SKS	'1134	Device SKS 11
SKS	'1234	Device SKS 12
SKS	'1334	Device SKS 13
SKS	'1434	Device SKS 14
SKS	'1534	Device SKS 15
SKS	'1634	Device SKS 16
SKS	'1734	Device SKS 17

Card Reader - Model 316/516-61

OCP	'0005	Read One Hollerith Card
OCP	'0105	Read One Binary Card
SKS	'0005	Skip if Card Reader Ready
SKS	'0105	Skip if Card Reader Not Busy
SKS	'0205	Skip if Not End of File
SKS	'0305	Skip if Card Reader Operational
SKS	'0405	Skip if Card Reader Not Interrupting
INA	'0005	Input From Card Reader if Ready
INA	'1005	Clear A-Register and Input From Card Reader if Ready
SMK	'0020	Set Interrupt Mask (A_{12})

Line Printer - Model 316/516-7050

OCP	'0003	No paper advance
OCP	'0203	Advance paper to channel 2
OCP	'0303	Allow memory scan via DMA/DMC
OCP	'0403	Advance paper to channel 1
OCP	'0703	Allow memory scan via the I/O Bus
SKS	'0003	Skip if ready
SKS	'0203	Skip if no alarm
SKS	'0303	Skip if odd column next
SKS	'0403	Skip if not interrupting
SKS	'1103	Skip if line is printed

SKS	'1203	Skip if not shuttling
SKS	'1303	Skip if line is printed and not shuttling
SKS	'1403	Skip if not advancing paper
SKS	'1503	Skip if line is printed and not advancing paper
SKS	'1603	Skip if not shuttling and not advancing paper
SKS	'1703	Skip if not busy
OTA	'0003	Output to line printer if ready
SMK	'0020	Set Interrupt Mask (A_{14})

Magnetic Tape Systems - Model 316/516-4100

OCP	1001X	Read BCD, 2 char/word
OCP	'011X	Read binary, 2 char/word
OCP	'021X	Read binary, 3 char/word
OCP	'031X	Set up Normal DMC/DMA mode
OCP	'041X	Write BCD, 2 char/word
OCP	'051X	Write binary, 3 char/word
OCP	'061X	Write end of file
OCP	'071X	Reset DMC/DMA mode
OCP	'101X	Write binary, 3 char/word
OCP	'111X	Space forward one space
OCP	'121X	Space forward one file
OCP	'131X	Set up DMC/DMA in Auto Switch mode
OCP	'141X	Rewind
OCP	'151X	Backspace one record
OCP	'161X	Backspace one file
OCP	'171X	Stop write
SKS	'001X	Skip if ready
SKS	'011X	Skip if not busy
SKS	'021X	Skip if an error has not been detected
SKS	'031X	Skip if not at beginning of tape (loadpoint)
SKS	'041X	Skip if not interrupting
SKS	'051X	Skip if end of tape has not been detected
SKS	'061X	Skip if end of file has not been detected
SKS	'071X	Skip if writing is permitted
SKS	'111X	Skip if MTT operational
SKS	'121X	Skip if DMA/DMC subchannel is not currently processing Channel 2
SKS	'131X	Skip if DMC/DMA subchannel is not in Auto Switch mode
SKS	'141X	Skip if not rewinding
INA	'001X	Input from TCU if ready
INA	'101X	Clear A-register and input from TCU if ready
OTA	'001X	Output data to the TCU
SMK	'0020	Set TCU Interrupt Mode, (A_1) for TCU 1, (A_2) for TCU 2

Fixed Head Disc File - Model 316/516-4400

INA	'0022	Input to A-register
INA	'1022	Clear A-register and Input to A-register
OTA	'0022	Output from A-register
OCP	'0322	Select DMA or DMC operation
OCP	'0422	Stop data transfer/acknowledge interrupt
OCP	'0722	Select I/O bus operation
SKS	'0022	Skip if Fixed Head Disc File ready
SKS	'0122	Skip if Fixed Head Disc File is not busy
SKS	'0222	Skip if Fixed Head Disc File has not detected a data transfer error
SKS	'0322	Skip if Fixed Head Disc File has not detected an access error
SKS	'0422	Skip if Fixed Head Disc File is not interrupting
SMK	'0020	Set Interrupt Mask (A_8)

Moving Head Disc File - Model 316/516-4600

INA	'1025	Clear A-register and input to A-register
INA	'0025	Input to A-register
OTA	'0025	Output from A-register
OCP	'0025	Return to zero seek
OCP	'0125	Direct seek
OCP	'0225	Read current address
OCP	'0325	Enable DMC/DMA mode of data transfer
OCP	'0525	Write track format
OCP	'0625	Read/write record
OCP	'0725	Enable I/O bus mode of data transfer
OCP	'1025	Stop transfer
OCP	'1425	Acknowledge interrupt
SKS	'0025	Skip if ready
SKS	'0125	Skip if not busy
SKS	'0225	Skip if data error not detected
SKS	'0325	Skip if setup error not detected
SKS	'0425	Skip if not interrupting
SKS	'1425	Skip if unit 1 not seeking
SKS	'1525	Skip if unit 2 not seeking
SKS	'1625	Skip if unit 3 not seeking
SKS	'1725	Skip if unit 4 not seeking
SMK	'0020	Set interrupt mask (A_4)

Process Interface Controller - Model 516-8100 Series

OCP	'XX33	Acknowledge designated subsystem interrupt
SKS	'0023	Skip if PIC is ready
SKS	'0033	Skip if PIC is not interrupting
SKS	'XX33	Skip if designated subsystem is not interrupting
INA	'0023	Input from PIC Adapter if ready
INA	'1023	Clear Register A and input from PIC Adapter if ready
OTA	'0123	Output select to PIC if ready
OTA	'0323	Output data to PIC if ready
OTA	'0723	Output and cycle PIC if ready
OTA	'1033	Set PIC interrupt mask

Single Line Controller (Honeywell 316/516)

OCP	'0060*	Enable receiver
OCP	'0160	Receive sync (synchronous controller only)
OCP	'0260	Enable transmitter
OCP	'0360	Set data terminal ready
OCP	'0460	Originate call
OCP	'0560	Enable DMC receive mode
OCP	'0660	Enable DMC transmit mode
OCP	'0760	Enable low speed receiver
OCP	'1060	Disable receiver
OCP	'1160	Transmit break (asynchronous controller only)
OCP	'1260	Disable transmitter
OCP	'1360	Reset data terminal ready
OCP	'1560	Disable DMC receive mode
OCP	'1660	Disable DMC transmit mode
OCP	'1760	Enable low speed transmitter
SKS	'0060	Skip if receiver ready
SKS	'0160	Skip if receiver fault is set
SKS	'0260	Skip if transmitter ready
SKS	'0360	Skip if no ring signal
SKS	'0460	Skip if controller not interrupting
SKS	'0560	Skip if no receiver ERL signal
SKS	'0660	Skip if no transmitter ERL signal
SKS	'1060	Skip if receiver ready
SKS	'1260	Skip if transmitter not busy (synchronous controller only)
SKS	'1360	Skip if no disconnect signal
SKS	'1460	Skip if no abandon call and retry signal
OTA	'0160	Skip if receiver fault is set and reset it
OTA	'0260	Transmit character

*'60 is address of first channel.

INA	'0060	Input character
INA	'1060	Clear A-register and input character
SMK	'0420	

Table below shows addresses of the first four channels and mask bit assignments.

	Address	Mask Bit
Single line controller 1	$60_{(8)}$	1
Single line controller 2	$61_{(8)}$	2
Single line controller 3	$62_{(8)}$	3
Single line controller 4	$63_{(8)}$	4

APPENDIX F
DEDICATED LOCATIONS

Octal Address	Assignment
00000	Index Register
00001 thru 00017	Protected Fill Program
00020	Starting Addresses DMC Channel 1
00021	Final Addresses DMC Channel 1
00022 thru 00057	DMC Channels 2 thru 16
00060	Power Failure Interrupt Link
00061	Real-Time Clock
00062	Memory Lockout Violation Int. Link
00063	Standard Interrupt Link
00064	Optional PI No. 1 Link
00065 thru 00143	Optional PI No. 2 thru 48 Links

APPENDIX G
KEY-IN LOADER

Key-In Loader for ASR-33/35/High-Speed Paper Tape Reader

```
*     TO LOAD PAL-MODE PROGRAMS, THE FOLLOWING PROCEDURE
*     MUST BE FOLLOWED.
*       1.  IF THE KEY-IN LOADER IS NOT PRESENT IN LOCATIONS
*           1-17 OCTAL, MANUALLY KEY IN THE FOLLOWING:
*                                      ASR       DIGITRONICS
*               1 STA    '57           010057    010057
*               2 OCP    '0001/4       030004    030001
*               3 INA    '10001/4      131004    131001
*               4 JMP    *-1           002003    002003
*               5 SNZ                  101040    101040
*               6 JMP    *-3           002003    002003
*               7 STA    0             010000    010000
*               10 INA   '1001/4       131004    131001
*               11 JMP   *-1           002010    002010
*               12 LGL   8             041470    041470
*               13 INA   '0001/4       130004    130001
*               14 JMP   *-1           002013    002013
*               15 STA*  0             110000    110000
*               16 IRS   0             024000    024000
*               17 SZE                 100040    100040
*
*       2.  MASTER CLEAR
*       3.  SET P REGISTER TO 1
*       4.  MOUNT PAL-MODE TAPE IN INPUT DEVICE AND PRESS START
```

Expansion to Key-In Loader to Clear Memory

```
 0 OCT  22        000022
20 EXA            000013
21 JMP 15         002015
```

Master Clear and Start at location 20.

APPENDIX H
SUMMARY OF DAP-16 PSEUDO-OPERATIONS

Oper. Mnemonic	Meaning	Class	Contents of Fields			Effect
			Location	Operation	Variable	
***	ZERO op-code	Special mnemonic code	Normal	***	Normal	Zeros put into op code
ABS	Absolute mode	Assembly control	Not Applicable	ABS	Ignored	Subsequent instructions assembled in absolute mode
BCI	Binary coded information	Data defining	Normal	BCI	N, followed by 2N alphanumeric characters	2N alphanumeric characters (N<30) converted into binary
BES	Block ending with symbol	Storage allocation	Normal; assigned location counter value after increase	BES	Previously defined absolute expression	Increases value of location counter by value of expression in the variable field
BSS	Block starting with symbol	Storage allocation	Normal; assigned location counter value before increase	BSS	Previously defined absolute expression	Same as BES
BSZ	Block storage of zeros	Storage allocation	Normal	BSZ	Previously defined absolute expression	Same as BES (used for defining storage blocks that are initially cleared)
CALL	Call subroutine	Program linking	Normal	CALL	Name of a subroutine	Generates a JST* to call referenced subroutine through transfer vector
CFx	Configuration	Assembly control	Not Applicable	CF1 or CF3 or CF4 or CF5	Ignored	Specifies which DAP-16 class computer will execute the object program. CF1 for DDP-116 CF3 for H316 CF4 for DDP-416 CF5 for DDP-516
COMN	Put in common storage	Storage allocation	Normal	COMN	Previously defined absolute expression	Value of expression in variable field are used to assign location in a common data pool for symbol in location field; facilitates reference by other programs
DAC	Define address constant	Data defining	Normal	DAC	Previously defined absolute or relocatable expression	Causes DAP-16 to assemble a 16-bit address word
DBP	Double precision	Data defining	Normal	DBP	Decimal subfields	Decimal characters converted into binary with double precision option
DEC	Decimal-to-binary	Data defining	Normal	DEC	Decimal subfields	Decimal characters converted into binary
END	End of assembly pass	Assembly control	Not Applicable	END	Address for transfer of control, following loading process	Terminates assembly pass

APPENDIX H (Cont)
SUMMARY OF DAP-16 PSEUDO-OPERATIONS

Oper. Mnemonic	Meaning	Class	Contents of Fields			Effect
			Location	Operation	Variable	
EQU	Equals	Symbol defining	Normal (See "Effect" column)	EQU	Previously defined absolute or relocatable expression	Causes DAP to assign the value and mode of the expression in the variable field to the symbol in the location field
EXD	Enter extend mode	Louder control	Not Applicable	EXD	Ignored	Subsequent instructions assembled in extended addressing mode
FIN	Finish	Assembly control	Not Applicable	FIN	Ignored	Punch out literals
LOAD	Load mode	Assembly control	Not Applicable	LOAD	Ignored	Subsequent instructions assembled in load mode
LIST	Generate listing	List control	Not Applicable	LIST	Ignored	Causes printout of source and object programs, side-by-side
LXD	Leave extend mode	Loader control	Not Applicable	LXD	Ignored	Subsequent instructions assembled in normal addressing mode
MOR	More	Assembly control	Not Applicable	MOR	Address for transfer of control	Interrogate sense switches to determine type of assembly control
NLST	No listing	List control	Not Applicable	NLST	Ignored	Inhibits program printout
OCT	Octal-to-binary	Data defining	Normal	OCT	Octal subfields	Octal characters converted into binary
ORG	Origin	Assembly control	Normal	ORG	Previously defined absolute or relocatable expression	Value and mode of expression in variable field is equivalent and location counter is set accordingly
PZE	Plus zero	Special mnemonics code	Normal	PZE	Normal	Zeros put into op code
REL	Relocatable mode	Assembly control	Not Applicable	REL	Ignored	Subsequent instructions assembled in relocatable mode
SETB	Set base mode	Loader control	Normal	SETB	Previously defined absolute or relocatable expression	Specify a sector other than zero as the base sector
SUBR	Entry point	Program linking	Ignored	SUBR	Name of subroutine, entry address	Punches subroutine name for identification in subroutine library
XAC	External address constant	Program linking	Normal	XAC	Name of subroutine	Causes DAP-16 to assemble 16-bit address word defining location outside the program

APPENDIX I
SOFTWARE PACKAGE

Tables that list all routines in the Honeywell 316/516 software package, their document number, the format, and equipment required for each routine are listed in this appendix. Utility routines are given in Table I-1, Input/Output routines are given in Table I-2, mathematical routines are given in Table I-3, and Test and Verification routines are given in Table I-4.

Table I-1.
Utility Routines

Type and Function	Mnemonic	Doc. No.	Format*	Equipment** Required
Assemble DAP-coded source program	DAP-16	180275000		
Chain or segment program	CHAIN	180070000		
Check:				
Error entry of halt	FER, FHT	182602000		
Overflow (and set error flag)	OVERFL	182600000		
Pseudo sense lights on/off	SLITE			
Pseudo sense lights	SLITET	182599000		
Sense switches	SSWTCH			
Compile FORTRAN-coded source program	FRTN	180463000		8K memory minimum
Convert indirect address to direct address	ARG$	180072000		
Debug (search, modify, clear memory, enter breakpoints)	DEBUG	180430000		
Convert decimal no. S to octal equivalent	DEC-OCT	180575000		
FORTRAN Debugging Aid (Trace)	F$TR	180073000		
DAP/FORTRAN loaders				
Expanded loader:				
ASR Input (paper tape) or	LDR-APM	180005000	DAP self-loading and object	
Paper Tape Reader Input or	LDR-APM			Paper Tape Reader
Magnetic Tape or Disc	LDR-APM			Magnetic Tape or Disc

Table I-1. (Cont)
Utility Routines

Type and Function	Mnemonic	Doc. No.	Format*	Equipment** Required
Card Reader Input	LDR-C	180582000		Card Reader
Standard Loaders:				
ASR Input (Paper Tape)	SLDR-A	180341000	Self-loading and DAP object	
Paper Tape Reader	SLDR-P	180342000	Self-loading and DAP object	Paper Tape Reader
ASR or Paper Tape Reader or Card Reader Input	MINILOAD	180580000		
Card Reader Input	SLDR-C	180583000		Card Reader
Dump:				
ASR Typewriter	DUMP	188806000		
High Speed Punch	X-16 DUPE	180087000		High Speed Punch
Line Printer	LP-DMP-5	180614000		Line Printer
Logic:				
Logical Complement	N$33	180090000		
Logical or	L$33	180065000		
Object Program Punch and Load	PAL-AP	180311000	Self-loading and DAP object	
Transfer arguments from calling to called routine	F$AT	180071000		
Update:				
Symbolic source update	SSUP	180767000		
Symbolic source update, I/O supervisor	SSUP-IOS	180000000		Paper Tape reader and punch, or two magnetic tape transports
Symbolic source update, revised dummy selection	SSUP RDS	180304000		

* All routines are in DAP format unless otherwise specified.

** "Equipment Required" is basic (ASR-33 or ASR-35 I/O) unless otherwise specified.

Table I-2.
Input/Output Routines*

Type and Function	Mnemonic	Doc. No.
FORTRAN IV Drivers:		
ASR Typewriter -		
Input	F$R1	182610000
Output	F$W1	182611000
Paper Tape Reader	F$R2	182612000
Paper Tape Punch	F$W2	182613000
Card Reader	F$R3	182614000
Line Printer	F$W4	182616000
Advance + Print Control	O$LP	180770000 **
Magnetic Tape Transport		
Input	F$R5-9	180306000
Output	F$W5-9	180307000
Write File Mark	F$D5-9	180308000
Rewind	F$B5-9	180309000
Back Space	F$F5-9	180310000
Device n		
Input	F$RN	180088000
Output	F$WN	180089000
FORTRAN IV:		
Format Control	F$IO	182618000
Argument Transfer	F$AR	
Buffer Closeout	F$CB	
DAP I/O Supervisors (Expanded)		
Selectable I/O	IOS-516X	180324000
Selectable I/O and Disc	IOS-516D	180278000
DAP I/O Supervisors (Preselected I/O Devices)		
ASR only	IOS-5AAA	180323000
High Speed Reader, High Speed Punch, ASR	IOS-5RPA	180573000
High Speed Reader and ASR	IOS-5RAA	180592000
Card Reader and ASR	IOS-5CAA	180618000
Card Reader, High Speed Punch, ASR	IOS-5CPA	180594000
Standard Library:		
ASR Typewriter -		
Type a line	O$AP	180255000
Carriage return	O$AC	
Advance to next line	O$AF	
Initialize heading	O$HH	180774000
Initialize listing	O$LL	
ASR Paper Tape Reader - or Keyboard		
ASCII	I$AA	189001000
Binary	I$AB	189002000
ASR Paper Tape Punch		
ASCII	O$AA	189003000
Binary	O$AB	189004000
Leader	O$AL	189005000

* All routines are in DAP object format.
** For use on the DDP-516 only.

Table I-2. (Cont)
Input/Output Routines*

Type and Function	Mnemonic	Doc. No.
High Speed Paper Tape Reader -		
ASCII	I$PA	189006000
Binary	I$PB	189007000
High Speed Paper Tape Punch -		
ASCII	O$PA	189008000
Binary	O$PB	189009000
Listing	O$PL	181479000
Heading	O$PH	
Leader	O$PLDR	189008000
Punch one line	O$PP	
Punch carriage return	O$PC	180257000
Advance to next line	O$PF	
Card Reader		
ASCII	I$CA	180110000
Binary	I$CB	180609000
Line Printer	O$LA	180768000**
	O$LB	180768000
Magnetic Tape		
Input -		
BCD (2 characters/wd)	I$MA-U	
Binary (2 characters/wd)	I$MB	180599000
Binary (3 characters/wd)	I$MC	
Output -		
BCD (2 characters/wd)	O$MA-U	
Binary (2 characters/wd)	O$MB	180598000
Binary (3 characters/wd)	O$MC	
File Mark	O$ME	
Backspace		
One file -	C$BF	
One record	C$BR	
Rewind	C$MR	182606000
Forward space -		
One file	C$FF	
One record	C$FR	
Conversion -		
ASCII code to IBM tape code	C$8T06	180082000
IBM tape code to ASCII code	C$6T08	180091000
Translate transport numbers	M$UNIT-U	180602000
Disc		
Format	M$FT	180666000
Fixed Head	D$IO	180617000
Moving Head	M$IO	180616000

* All routines are in DAP object format.
** For use on the Honeywell 416/516 only.

Table I-3.
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
Complex:			
Absolute value	CABS	182596000	FTRN object
Add	A$55	182544000	FTRN object
Add single precision argument	A$52	180041000	FTRN object
Conjugate	CONJG	182598000	FTRN object
Convert imaginary part to real	AIMAC	182578000	DAP object
Cosine	CCOS	180066000	FTRN object
Divide	D$55	180034000	FTRN object
Divide by single precision argument	D$52	180044000	FTRN object
Exponential, base e	CEXP	182593000	FTRN object
Load	L$55	182542000	DAP object
Logarithm, base e	CLOG	182591000	FTRN object
Multiply	M$55	182545000	FTRN object
Multiply by single precision argument	M$52	180045000	FTRN object
Negate a complex quantity	N$55	180069000	FTRN object
Raise to integer power	E$51	182594000	FTRN object
Sine	CSIN	182595000	FTRN object
Square root	CSQRT	182592000	FTRN object
Store (hold)	H$55	182543000	DAP object
Subtract	S$55	180093000	FTRN object
Subtract single precision argument	S$52	180042000	FTRN object
Double Precision:			
Fixed-Point:			
Add	DADD	188812000	DAP object
Arctangent	DATNX1	188793000	DAP object
Arctangent*	DATNX2	188794000	DAP object
Cosine	DCOSX1	188792000	DAP object
Cosine*	DCOSX2	180762000	DAP object
Divide	DDIV	188808000	DAP object
Divide*	DDIVH	188809000	DAP object
Exponential, base e	DEXEX1	188799000	DAP object
Exponential, base e*	DEXEX2	188800000	DAP object
Exponential, base 2	DEX2X1	188797000	DAP object
Exponential, base 2*	DEX2X2	188798000	DAP object

*Operates with High Speed Arithmetic Unit (Honeywell 316/516-11) option only.

Table I-3. (Cont)
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
Logarithm, base e	DLGEX1	188801000	DAP object
Logarithm, base e*	DLGEX2	188802000	DAP object
Logarithm, base 2	DLG2X1	188795000	DAP object
Logarithm, base 2*	DLG2X2	188796000	DAP object
Multiply	DMPY	188808000	DAP object
Multiply*	DMPYH	188809000	DAP object
Round up binary number	RODD	188804000	DAP object
Sine	DSINX1	188790000	DAP object
Sine*	DSINX2	188791000	DAP object
Square Root	DSQRX1	188788000	DAP object
Square Root*	DSQRX2	188789000	DAP object
Subtract	DSUB	188813000	DAP object
Two's complement	TWOS	188803000	DAP object
Floating-Point:			
Absolute value	DABS	182587000	FTRN object
Add	A$66	182540000	DAP object
Add*	A$66X	180680000	DAP object
Add single precision argument	A$62	180037000	FTRN object
Add integer to exponent	A$81	180064000	DAP object
Arctangent, principal value	DATAN	182584000	FTRN object
Arctangent, x/y	DATAN2	180056000	FTRN object
Clear (zero) exponent	Z$80	180060000	DAP object
Convert exponent to integer	C$81	180046000	DAP object
Convert to integer	C$61	182554000	DAP object
Convert to single-precision (from pseudo accumulator)	C$62	182576000	DAP object
Cosine	DCOS	189955999	FTRN object
Divide	D$66	182541000	DAP object
Divide by single precision argument	D$62	180040000	FTRN object
Exponential, base e	DEXP	182581000	FTRN object
Load	L$66	182538000	DAP object
Logarithm, base e	DLOG	182579000	FTRN object
Logarithm, base 2	DLOG2	182579000	FTRN object
Logarithm, base 10	DLOG 10	180051000	FTRN object
Maximum value	DMAX1	182585000	DAP object
Minimum value	DMIN1	182586000	DAP object
Multiply	M$66	182541000	DAP object

*Operates with High Speed Arithmetic Unit option only.

Table I-3. (Cont)
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
Multiply by single precision argument	M$62	180039000	FTRN object
Negate	N$66	180061000	DAP object
Raise to double precision power	E$66	180054000	FTRN object
Raise to integer power	E$61	180052000	FTRN object
Raise to single precision power	E$62	180053000	FTRN object
Remainder	DMOD	182588000	FTRN object
Sine	DSIN	182583000	FTRN object
Square root	DSQRT	182580000	FTRN object
Store (hold)	H$66	182539000	DAP object
Subtract	S$66	182540000	DAP object
Subtract single precision argument	S$62	180038000	FTRN object
Transfer sign of second argument to first	DSIGN	182589000	FTRN object
Truncate fractional bits	DINT	180049000	DAP object
Integer:			
Absolute value	LABS	182552000	DAP object
Convert to double precision	C$16	180059000	FTRN object
Convert (FORTRAN-Generated) to single precision	FLOAT	180062000	DAP object
Convert to single precision	C$12	182575000	DAP object
Divide	D$11	182546000	DAP object
Divide*	D$11X	180686000	DAP object
Maximum single precision value	AMAXO	182548000	DAP object
Maximum value	MAXO	182548000	DAP object
Multiply	M$11	180035000	DAP object
Multiply*	M$11X	180685000	DAP object
Positive difference	IDIM	182556000	DAP object
Raise to integer power	E$11	182547000	DAP object
Raise to integer power*	E$11X	180684000	DAP object
Remainder	MOD	182555000	DAP object
Transfer sign of second argument to first	ISIGN	182557000	DAP object
Single Precision:			
Fixed-point:			
Arctangent	ARNX1	188779000	DAP object
Arctangent*	ATNX2	188780000	DAP object

* Operates with High Speed Arithmetic Unit option only.

Table I-3. (Cont)
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
Cosine	COSX1	188781000	DAP object
Cosine*	COSX2	180761000	DAP object
Divide	DIV	188810000	DAP object
Exponential, base e	EXEX1	188786000	DAP object
Exponential, base e*	EXEX2	188787000	DAP object
Exponential, base 2	EX2X1	188782000	DAP object
Exponential, base 2*	EX2X2	188783000	DAP object
Logarithm, base e	LGEX1	188814000	DAP object
Logarithm, base e*	LGEX2	188815000	DAP object
Logarithm, base 2	LG2X1	188784000	DAP object
Logarithm, base 2*	LG2X2	188785000	DAP object
Multiply	MPY	188811000	DAP object
Round up binary number	ROND	188805000	DAP object
Sine	SINX1	188777000	DAP object
Sine*	SINX2	188778000	DAP object
Square root	SQRX1	188775000	DAP object
Square root*	SQRX2	188776000	DAP object
Floating-Point:			
Absolute Value	ABS	182570000	DAP object
Add	A$22	182536000	DAP object
Arctangent, principal value	ATAN	182564000	DAP object
Arctangent, y/x	ANTAN2	182564000	DAP object
Convert (FORTRAN-generated) to double precision	DBLE	180058000	DAP object
Convert to integer or truncate fractional bits and convert to integer	IFIX	182553000	DAP object
Convert pair to complex	CMPLX	182597000	FTRN object
Convert to complex format	C$25	180068000	FTRN object
Convert to double precision	C$26	182590000	DAP object
Convert to integer	C$21	182558000	DAP object
Divide	D$22	182537000	DAP object
Exponential, base e	EXP	192561000	DAP object
Hyperbolic tangent	TANH	182565000	DAP object
Load	L$22	182534000	DAP object
Logarithm, base e	ALOG	182559000	DAP object
Logarithm, base e*	ALOGX	180682000	DAP object

*Operates with High Speed Arithmetic Unit option only.

Table I-3. (Cont)
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
Logarithm, base 10	ALOG10	182559000	DAP object
Maximum integer value	MAX1	182549000	DAP object
Maximum value	AMAX1	182549000	DAP object
Minimum integer value	MIN1	182551000	DAP object
Minimum value	AMIN1	182551000	DAP object
Multiply	M$22	182537000	DAP object
Multiply*	M$22X	180683000	DAP object
Positive difference	DIM	182573000	DAP object
Raise to double precision power	E$26	182582000	FTRN object
Raise to integer power	E$21	182562000	DAP object
Raise to single precision power	E$22	180045000	DAP object
Remainder	AMOD	182572000	DAP object
Sine	SIN	182563000	DAP object
Square root	SQRT	182560000	DAP object
Square root*	SQRTX	180681000	DAP object
Store (hold)	H$22	182535000	DAP object
Subtract	S$22	182536000	DAP object
Transfer sign of second argument to first	SIGN	182536000	DAP object
Truncate fractional bits	AINT	182571000	DAP object
Two's complement	N$22	180097000	DAP object

*Operates with High Speed Arithmetic Unit option only.

Table I-4.
Test and Verification Routines

Type and Function	Name	Doc. No.	Format	Option Required
Central Processor Test No. 3	X16-CCT3	70180658000	PAL	316/516
High Speed Arithmetic Test	X16-11T1	70180294000	PAL	316/516-11
Core Memory Test (DDP-516)	X16-CMT1	70180265000	PAL	STD-516
Core Memory Test (H316)	316-CMT1	70180773000	PAL	STD-316
Memory Bank Switching Test	516-05T1	70180316000	PAL	Over 24K Memory
Memory Lockout Test No. 1	X16-08T1	70180318000	PAL	516-08
Power Failure Test No. 2	X16-PFT2	70180608000	PAL	316/516
Teleprinter Test Program	X16-TLT1	70180269000	PAL	ASR-33/35
Card Reader Test	X16-CRT1	70180267000	PAL	516-61
Line Printer Test	X16-PRT1	70180264000	PAL	516-7050
Magnetic Tape Test	X16-MTT2	70180452000	PAL	516-4100
Fixed Head Disc Test	X16-44T1	70180454000	PAL	516-4400
Moving Head Disc Test	X16-46T1	70180713000	PAL	516-4600
Real-Time Clock Test	X16-RTC1	70180263000	PAL	316/516-12
Fixed Head Disc Test (SMS)	X16-43T1	70180834000	PAL	316/516-4300
33-35 Teletypewriter Test	X16-TWT1	70180654000	PAL	316/516
High Speed Paper Tape Reader/Punch Test	X16-RPT2	70180967000	PAL	316/516-50/52