DDP-516
USERS GUIDE

March 1967

Honeywell

CONTENTS

CONTENTS (Cont)

SECTION VI
UTILITY LIBRARY

SECTION VII
MATHEMATICAL LIBRARY

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

TABLES (Cont)

DDP-516 General Purpose Computer

SECTION I
INTRODUCTION

The User's Guide is intended to familiarize you with the operation of the DDP-516 computer. From a user's point of view, the DDP-516 computer is a relatively simple device to operate. Its movable control console is designed to allow complete operator freedom which provides for easy, error-free operation. The comprehensive software package supplied with the DDP-516 includes a FORTRAN IV compiler, a DAP-16 assembler, a variety of utility routines and an extensive subroutine library. Furthermore, with each DDP-516, Honeywell provides a complete library of instruction manuals and program listings that tell the programmer/operator all he needs to know about preparing and using FORTRAN IV and DAP-16 programs.

The FORTRAN IV and DAP-16 Manuals (3C Doc. No. 130071634 and 130071629) present complete instructions for preparing programs in the two DDP-516 source languages. The Programmers Reference Manual (3C Doc. No. 130071585) lists and describes all the instructions in the DDP-516 repertoire.

In the User's Guide, you will find detailed instructions for using your computer and its software package. The guide is sectionalized for easy referencing so that information you will refer to again and again can be found easily. At the same time, the user can progressively increase his knowledge of the DDP-516 by reading the sections in the order they appear in the guide. Sections II and III will be of greatest interest initially. Section II explains the function of each of the controls on the control console and presents a few simple procedures for using them to perform basic, manual operations. Section III lists, by type and function, all the programs in your software package and describes the formats in which they are punched on paper tape.

Section IV presents step-by-step instructions for generating self-loading system tapes that are tailored to your installation. These tapes will enable you to use your DDP-516 efficiently and conveniently. Section V contains generalized operating procedures for using your newly generated system tapes and for using some of the more frequently used utility programs. The remainder of the User's Guide contains reference material designed to help you make effective use of your DDP-516 and its software package.

SECTION II
CONTROL CONSOLE

The basic DDP-516 computer system includes a main frame, a control console and a typewriter. In addition to these standard devices, your installation may include any or all of the following peripheral devices:

High-speed paper tape reader	Line printer
High-speed paper tape punch	One or more magnetic tape units
Card-reader	One or two disc file units

The DDP-516 Programmers Reference Manual (3C Doc. No. 130071585) and the appropriate peripheral device option manuals contain operating instructions for each device, whether standard or optional. The software package contains the input/output subroutines required for using the devices included in your installation.

The control console (Figure 2-1) includes the controls and indicators for normal system operation. These controls and indicators are listed and described in Tables 2-1 through 2-3.
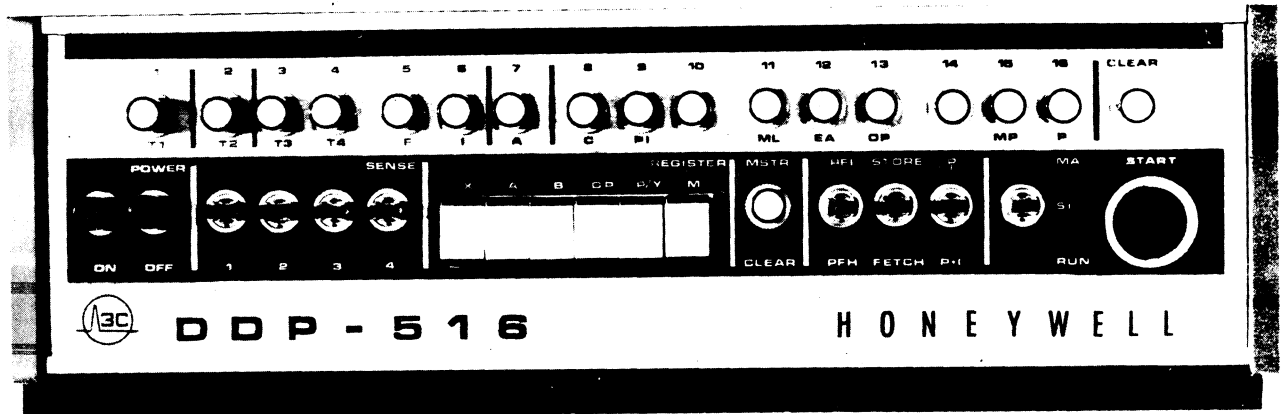
3877

Figure 2-1. DDP-516 Control Panel

Table 2-1.
Registers Displayed on DDP-516 Control Panel

Register	Bits	Display
X	1-16	Index Register
A	1-16	Register A (primary arithmetic and logic register).
B	1-16	Register B (secondary arithmetic and logic register).
OP	1-16	State of key flip-flops in system. (Refer to Table 2-4.)
P/Y	1-16	Register Y (memory address register). When the computer is operating in the memory access (MA) mode, Registers P and Y contain the same address. When in the single instruction (SI) or RUN mode Register P contains an address which is one greater than the address contained in Register Y.
M	1-16	Register M (memory buffer register) contains the contents of that memory location specified by Register Y.

Table 2-2.
Function of Control-Panel Pushbuttons

Pushbutton	Function
16 Indicator	Enter or display data. A "ONE" is indicated when pushbutton/indicator is illuminated. A "ZERO" is indicated when pushbutton/indicator is not illuminated.
CLEAR	Clears displayed registers (A, B, P/Y or M only).
POWER-ON	Applies power to main frame. The indicator is illuminated when power is applied.
POWER-OFF	Removes power from main frame.
X	Displays contents of index register. (Data can be entered into Register X via location 00000_8.)
A	Displays and permits alterations in contents of Register A.
B	Displays and permits alterations in contents of Register B.
OP	Displays state of key flip-flops in the system.
P/Y	Displays and permits alterations in contents of Register P and Y. (Registers P and Y contain the same address when the computer is in the memory access (MA) mode. Register P contains an address which is one higher than that contained in Register Y when the computer is in the single instruction (SI) or RUN modes.)
M	Displays and permits alterations in contents of Register M.
MSTR CLEAR	Clears Registers A, B, P, Y and M; sets all timing to that state existing following a HLT (halt). Initializes standard peripheral devices and options.
START	Starts machine in all modes. Indicator is illuminated when machine is in RUN mode.

Table 2-3.
Function of Control-Panel Selector Switches

Switch	Position	Function
SENSE 1-2-3-4		Enables altering the course of a program via switch interrogation during program operation.
	Up	Switches are set.
	Down	Switches are reset.
PFI/PFH (Power Fail Interrupt/ Power Fail Halt)	PFI	Allows the machine to cause a program interrupt when power fails.
	PFH	Allows the machine to halt when power fails.
STORE/FETCH	STORE	Enables data to be written into memory when the computer is in the MA mode.
	FETCH	Enables data to be read out of memory when the computer is in the MA mode.
P/P+1	P	Enables accessing a specific memory location when the computer is in the MA mode.
	P+1	Enables accessing consecutive memory locations when the computer is in the MA mode.
MA/SI/RUN (Memory Access/ Single Instruction/ Run)	MA	Enables data to be written into or read from a location in memory and disables the protection for locations $1-17_8$.
	SI	Enables step-by-step execution of a program.
	RUN	Enables normal operating mode.

Data within memory and several of the main frame registers can be monitored on the control panel. Data can also be manually entered into memory and several registers from the control panel. Procedures for initializing the system, reading out and entering data into memory and the main frame registers are provided in the following paragraphs.

SYSTEM INITIALIZATION

The system can be initialized by depressing the MASTER CLEAR pushbutton. This operation has no effect on the contents of memory or Register X. The MASTER CLEAR function is also performed when power is applied to the system via the POWER-ON pushbutton.

REGISTER DISPLAY

The contents of Register X, A, B, OP, P/Y, or M can be displayed by depressing the appropriate REGISTER pushbutton. The contents of the selected register is displayed on the 16 pushbutton/indicators. Refer to Table 2-4 for the significance of the OP display.

REGISTER LOAD

Data is entered into Register A, B, P/Y, or M as follows.

a.   Depress the appropriate REGISTER pushbutton.

b.   Depress the CLEAR pushbutton.

c.   Enter the desired data by depressing the appropriate pushbutton/indicators (1 through 16).

(Data can be entered into Register X by addressing location 000000_8 and entering the data into that location. For the procedure to enter data into memory, refer to the paragraph on Memory Load in this section.)

Table 2-4.
OP Display

Bit	Significance	Bit	Significance
1	T1, timing level 1	9	P1, permit interrupt
2	T2, timing level 2	10	Unassigned
3	T3, timing level 3	11	ML, memory lockout (option)
4	T4, timing level 4	12	EA, extended addressing (bank switching option)
5	F, fetch cycle	13	DP, double precision (option)
6	I, indirect cycle	14	Unassigned
7	A, execute cycle	15	MP, memory parity error
8	C, c-bit	16	P, I/O parity error

MEMORY DISPLAY

The contents of any memory location is displayed as follows.

a. Set MA/SI/RUN switch to MA.

b. Set FETCH/STORE switch to FETCH.

c. Set P/P+1 switch to P.

d. Depress REGISTER-P/Y pushbutton.

e. Depress CLEAR pushbutton.

f. Depress the appropriate pushbutton/indicators (1 through 16) to designate the octal address of the memory location containing the data to be displayed.

g. Depress REGISTER-M pushbutton.

h. Depress START pushbutton. The contents of the addressed memory location will be displayed on the pushbutton/indicators (1 through 16).

The contents of successive memory locations can be displayed as follows.

i. Set P/P+1 switch to P+1.

j. Depress the START pushbutton. Each time the START pushbutton is depressed the contents of the next memory location is displayed.

MEMORY LOAD

Data is entered into any memory location as follows.

a. Set MA/SI/RUN switch to MA.

b. Set FETCH/STORE switch to STORE.

c. Set P/P+1 switch to P.

d. Depress REGISTER-P/Y pushbutton.

e. Depress CLEAR pushbutton.

f. Depress the appropriate pushbutton/indicators to designate the octal address of the memory location to be loaded.

g. Depress REGISTER-M pushbutton.

h. Depress CLEAR pushbutton.

i. Depress the appropriate pushbutton/indicators to enter the desired data into the addressed location.

j. Depress START pushbutton.

The desired data is now in the addressed location. Successive memory locations can be loaded as follows.

k. Set the P/P+1 switch to P+1.

l. Repeat steps h through j for each successive memory location to be loaded.

SINGLE INSTRUCTION OPERATION

A program is executed in the single-instruction mode as follows.

a. Set MA/SI/RUN switch to SI. (When the switch is not in the MA position, the FETCH/STORE and P/P+1 switches are disabled.)

b. Depress MASTER CLEAR pushbutton.

c. Enter initial parameters into Register A, B, or P/Y as required. (Refer to the procedure for Register Load described in this section.)

d. Depress START pushbutton.

The first instruction is fetched from memory and placed in Register M and may be examined by depressing the Register-M pushbutton. Thereafter, each time the START pushbutton is depressed, the previously fetched instruction is executed, the next instruction is fetched, and the computer halts. If the P/Y pushbutton is depressed, the address from which the new instruction was fetched is displayed on the pushbutton/indicators (1 through 16). During execution (run operation) the SI position may be used at any time to aid in program debugging.

RUN OPERATION

A program is executed in the run mode as follows.

a. Set MA/SI/RUN switch to RUN.

b. Depress MASTER CLEAR pushbutton.

c. Enter initial parameters into Register A, B, or P/Y as required. (Refer to the procedure for Register Load described in this section.)

d. Depress START pushbutton.

The program will run until a HALT is executed or until the MA/SI/RUN switch is set to SI.

KEY-IN LOADER

The octal instructions listed below normally occupy memory locations 1_8 through 17_8 and enables the loading of "self-loading" paper tapes via the teletype or high-speed paper tape reader.

Octal Address	Octal Instruction	Meaning
1	010057	STA '57
2	03000X	OCP '000X
3	13100X	INA '100X
4	002003	JMP *-1
5	101040	SNZ
6	002003	JMP *-3
7	010000	STA 0
10	13100X	INA '100X
11	002010	JMP *-1
12	041470	LGL 8
13	13000X	INA '000X
14	002013	JMP *-1
15	110000	STA *0
16	024000	IRS 0
17	100040	SZE

The value of "X" in the above instructions is dependent upon the input device used to read the paper tape. A "1" specifies the high-speed paper tape reader and a "4" specifies the teletype.

The hardware protects memory locations 1_8 through 17_8 from modification by a stored program. Therefore, under normal conditions, the key-in loader should remain intact in these locations. However, when operating in the MA mode, locations 1_8 through 17_8 are unprotected, therefore care must be exerted to avoid inadvertently destroying the key-in loader while loading memory. The key-in loader is entered into memory as follows.

a. Depress MSTR CLEAR pushbutton. The program counter (Register P) is set to zero.

b. Set MA/SI/RUN switch to MA. This unlocks the protected area, addresses $1-17_8$.

c. Set STORE/FETCH switch to store.

d. Set P/P+1 switch to P+1.

e.   Enter the first instruction (010057_8) into Register M.

f.   Depress START pushbutton. Register P will be incremented by one, and the first instruction will be entered into location 000001_8.

g.   Repeat steps f. and g. for each of the remaining instructions to be loaded. Each time the START pushbutton is depressed, Register P will be incremented by one, and the instruction in Register M will be loaded into memory.

Checking the Key-In Loader

The following instructions are performed to ensure that the key-in loader has been loaded correctly into the designated memory locations.

a.   Depress MSTR CLEAR pushbutton.

b.   Set MA/SI/RUN switch to MA.

c.   Set STORE/FETCH switch to FETCH.

d.   Set P/P+1 switch to P+1.

e.   Depress Register-M pushbutton.

f.   Depress START pushbutton. Register P will be incremented by one and the contents of memory location 000001_8 (010057) will be displayed on the pushbutton/ indicators (1 through 16).

g.   Repeat step f. for each of the remaining memory locations to be monitored. Each time the START pushbutton is depressed, Register P will be incremented by one and the contents of the addressed memory will be displayed.

SECTION III
SOFTWARE PACKAGE

This section presents tables that list all routines in the DDP-516 software package, their document number, the format, and equipment required for each routine. Utility routines are given in Table 3-1, Input/Output routines are given in Table 3-2, and Mathematical routines are given in Table 3-3.

Table 3-1.
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 or 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)	COP	188807000		
DAP/FORTRAN loaders				
Expanded loaders:				
ASR input (paper tape)	LDR-A	180335000	DAP self-loading & object	
Paper tape reader input	LDR-P	180336000	DAP self-loading & object	Paper Tape Reader

* All routines are in DAP object format unless otherwise specified.

** "Equipment Required" is basic (ASR-33 or ASR-35 I/O) unless otherwise specified.

Table 3-1. (Cont)
Utility Routines

Type and Function	Mnemonic	Doc. No.	Format*	Equipment** Required
Standard loaders:				
ASR input (paper tape)	SLDR-A	180341000	DAP self-loading	
Paper tape reader input	SLDR-P	180342000	DAP self-loading	Paper tape Reader
Dump:				
ASR Typewriter (various formats)	DUMP	188806000		
Logic:				
Logical complement	N$33	180090000		
Logical OR	L$33	180065000		
Object program punch and load	PAL-AP	180311000		
Transfer arguments from calling to called routine	F$AT	180071000		
Update:				
Symbolic source update	SSUP	180767000		
Symbolic source update, I/O supervisor	SSUP-IOS	180000000		Paper tape Reader and Punch, Magnetic Tape Transport
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 3-2.
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$R3	182614000
Magnetic Tape Transport		
Input	F$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$IO	
Argument Transfer	F$AR	182618000
Buffer Closeout	F$CB	
I/O Supervisors:		
DAP ASR	IOS-16A	180323000
DAP Expanded I/O	IOS-16B	180324000
FORTRAN IV	F4-IOS	180016000
Standard Library:		
ASR Typewriter -		
Type a line	O$AP	
Carriage return	O$AC	180255000
Advance to next line	O$AF	
Initialize heading	O$HH	180774000
Initialize listing	O$LL	
ASR Paper Tape Reader -		
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.

Table 3-2. (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
Magnetic Tape		
Input -		
BCD	I$MA	
Binary	I$MB	} 182604000
Binary (3 characters/word)	I$MC	
Output -		
BCD	O$MA	
Binary	O$MB	} 182605000
Binary (3 characters/word)	O$MC	
File mark	O$ME	
Backspace		
One file -	C$BF	
One record	C$BR	
Rewind	C$MR	} 182606000
Forwardspace -		
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	180228000

* All routines are in DAP object format.

Table 3-3.
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	AIMAG	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	1.80762000	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 Multiply/Divide option only.

Table 3-3. (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 single-precision argument	A$62	180037000	FTRN object
Add integer to exponent	A$81	180064000	DAP object
Arctangent, principle 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	DLOG10	180051000	FTRN object
Maximum value	DMAX1	182585000	DAP object
Minimum value	DMIN1	182586000	DAP object
Multiply	M$66	182541000	DAP object

* Operates with Multiply/Divide option only.

Table 3-3. (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	IABS	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
Maximum single-precision value	AMAXO	182548000	DAP object
Maximum value	MAXO	182548000	DAP object
Multiply	M$11	180035000	DAP object
Positive difference	IDIM	182556000	DAP object
Raise to integer power	E$11	182547000	DAP object
Remainder	MOD	182555000	DAP object
Transfer sign of second argument to first	ISIGN	182557000	DAP object
Single-precision:			
Fixed-point:			
Arctangent	ATNX1	188779000	DAP object
Arctangent*	ATNX2	188780000	DAP object
Cosine	COSX1	188781000	DAP object
Cosine*	COSX2	180761000	DAP object

* Operates with Multiply/Divide option only.

Table 3-3. (Cont)
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
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, principle 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	182561000	DAP object
Hyperbolic tangent	TANH	182565000	DAP object
Load	L$22	182534000	DAP object
Logarithm, base e	ALOG	182559000	DAP object
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

* Operates with Multiply/Divide option only.

Table 3-3. (Cont)
Mathematical Routines

Type and Function	Mnemonic	Doc. No.	Format
Multiply	M$22	182537000	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
Store (hold)	H$22	182535000	DAP object
Subtract	S$22	182536000	DAP object
Transfer sign of second argument to first	SIGN	182574000	DAP object
Truncate fractional bits	AINT	182571000	DAP object
Two's complement	N$22	180097000	DAP object

SECTION IV
GENERATION OF SELF-LOADING SYSTEM TAPES

The DDP-516 Software package contains, in addition to an extensive library of object tapes, a few tapes that are self-loading. The self-loading tapes are the loader (LDR or SLDR) and the punch and load (PAL-AP) programs. (PAL-AP is also supplied in relocatable object form.) These two programs enable you to generate self-loading system tapes tailored to your installation. (A DAP or FORTRAN system with less than 8K of memory cannot generate system tapes.) This section contains detailed procedures for generating self-loading system tapes for the DAP-16 Assembler, the FORTRAN IV (FRTN) Compiler and the Symbolic Source Update (SSUP) Program.

As illustrated in Figure 4-1, the procedure for generating a self-loading system tape (SLST) involves six operations. These operations are as follows.

a.   Entering the loader (LDR or SLDR) into high sectors of memory.

b.   Loading the main program (DAP-16, FRTN or SSUP) into memory with specified starting locations at which loading and cross-sector linkage are to begin.

c.   Loading the I/O routines (IOS, I/O Library, etc.) to satisfy the main program calls.

d.   Obtaining a memory map to determine the area of memory which contains data to be punched out on the system tape.

e.   Loading the punch and load program (PAL-AP) into the high sector of memory with the contents of a specified area of memory assigned to be punched out.

f.   Punching out the SLST.

The procedural steps that follow have been abbreviated so that the instruction "Load $XXXXX_8$ into Register _" implies the three step REGISTER LOAD procedure described in Section II. Therefore, it is recommended that you familiarize yourself with the control console before attempting to generate an SLST.

GENERATING A SELF-LOADING DAP-16 SYSTEM TAPE

The DAP-16 Manual (3C Doc. No. 130071629) and the DDP-516 Programmers Reference Manual (3C Doc. No. 130071585) contain a complete description of the DAP-16 symbolic assembly program and the instructions for preparing programs in DAP-16 source language. The following paper tapes from your library are required to generate a self-loading DAP-16 system tape.

a.   Some version of SLDR or LDR as determined from Table 4-1

b.   PAL-AP (self-loading form)

c.   DAP-16 (also contains IOS and MGTD)

d.   I/O Library

e.   DECCL and SETSIZ.
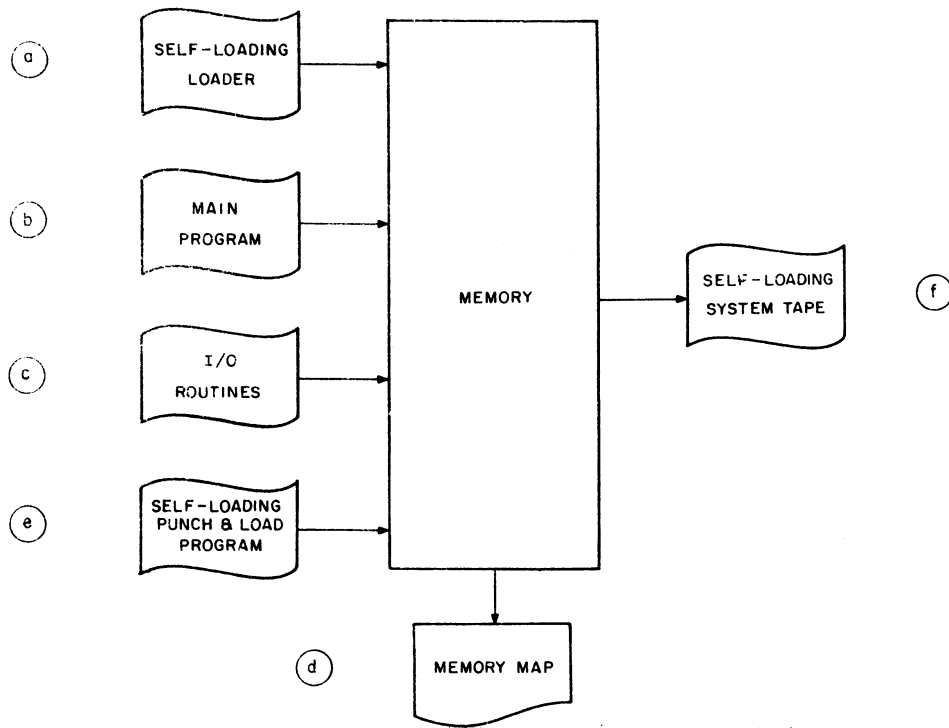
Figure 4-1. Self Loading System Tape Generation

Table 4-1.
Loader Selection

Device	Loader
Teletype:	
Standard Loader	SLDR-A
Expanded Loader	LDR-A
High Speed Paper Tape Reader:	
Standard Loader	SLDR-P
Expanded Loader	LDR-P

As indicated in Table 4-1, the peripheral devices at your installation determine which loader is required to generate the SLST. In addition, the number of passes desired during an assembly and the options available at your installation effect the choice of loader. The standard loaders do not allow a one pass assembly and will not function with systems using the extended addressing or memory lockout options. When you have determined which loader is required, proceed as directed in the following paragraphs.

Load the Self-Loading Loader

    a.   Check the key-in loader as described in Section II to ensure that it is intact. If the loader has been disturbed, you must reload it.

    b.   Depress the MSTR CLEAR pushbutton.

c. Load 000001_8 into Register P.

d. Insert the loader tape into the appropriate input device.

e. Set the MA/SI/RUN switch to RUN.

f. Depress the START pushbutton. (If you are loading from an ASR-33 teletype unit, the manual START switch on the device must be activated. If you are loading from an ASR-35 teletype unit, the MODE switch must be set to KT.) The loader program will be loaded into the high-order sectors of memory (the four high-order sectors for LDR or the three high-order sectors for SLDR), and the computer will halt.

Load the DAP-16 Object Tape

a. Depress the MSTR CLEAR pushbutton.

b. Load XX000, the starting location of the loader, into Register P (XX is the highest sector of memory).

c. Insert the DAP-16 object tape into the appropriate input device.

d. Depress the START pushbutton. The tape will be loaded into memory starting at location 400_8. Loading will halt before the entire tape is read and the teletype will produce MR to indicate that additional routines must be loaded.

e. Insert the I/O Library tape into the appropriate input device.

f. Depress the START pushbutton. The teletype will again produce MR.

g. Insert the tape containing DECCL and SETSIZ into the appropriate input device.

h. Depress the START pushbutton. The teletype will produce LC to indicate that loading is complete.

i. Load $XX002_8$ into Register P to obtain a memory map.

j. Depress the START pushbutton. A memory map, having the format shown in Table 4-2, will be produced on the teletype.

Table 4-2
Memory Map

* START	D D D D D	The entry location of the main program.
* HIGH	D D D D D	The first location not used by the main program.
* BASE	D D D D D	The first location not used in the base sector.
* NAMES	D D D D D	The lowest location used by the loader.
SUBI	D D D D D	The entry location of a subroutine.
.	.	.
.	.	.
.	.	.
LC	.	Loading complete

Punch the DAP System Tape

a. Depress the MSTR CLEAR pushbutton.

b. Load 000001_8 into Register P.

c. Insert the PAL-AP tape into the appropriate input device.

d. Depress the START pushbutton. The tape will be loaded into the highest 1200_8 locations of memory.

e. Load $XX100_8$ into Register P.

f. Load 100_8 into Register A. (If a teletype is being used, modify the Register-A load by setting bit 1 of the register. Depress the ON switch on the ASR-33 teletype punch or set the MODE switch to KT on the ASR-35 teletype unit.)

g. Depress the START pushbutton. The program will halt immediately.

h. Load into Register A location HIGH obtained from the memory map.

i. Depress the START pushbutton. The self-loading DAP-16 system tape will be punched out by the appropriate output device. To test your system tape, perform an assembly as described in Section V.

Computing DAP-16 Memory-Table Size

Each DAP-16 literal-, symbol-, and transfer-vector-table entry occupies three words in memory. To calculate the total number of locations available in your computer, subtract the HIGH address obtained from your DAP-16 memory map from the last address of your computer memory. Then subtract 100_8 from the remainder and divide the new remainder by 3. (All computations in octal.)

GENERATING A SELF-LOADING FORTRAN IV SYSTEM TAPE

The FORTRAN IV Manual (3C Doc. No. 130071364) contains instructions for the preparation of FORTRAN IV programs to be used on your DDP-516 computer. To generate a self-loading FORTRAN IV system tape, you will need the following tapes from your library.

a. Some form of SLDR or LDR as determined from Table 4-1.

b. PAL-AP (self-loading form)

c. FRTN

d. F4-IOS

When you have determined which loader is required, proceed as follows.

Load the FRTN Object Tape

a. Perform the procedure for loading the self-loading loader as described earlier in this section.

b. Depress the MSTR CLEAR pushbutton.

c. Load $XX000_8$, the starting location of the loader, into Register P (XX is the highest sector of memory).

d. Insert the FRTN object tape into the appropriate input device.

e. Depress the START pushbutton. (If you are loading from an ASR-33 teletype unit the manual START switch on the device must be activated. If you are loading from an ASR-35 teletype unit, the MODE switch must be set to KT.) When the tape has been read into memory, the teletype will produce MR to indicate additional routines must be loaded.

Load the FRTN I/O Selector

a. Insert the F4-IOS tape into the appropriate input device.

b. Depress the START pushbutton. When the tape has been read into memory, the teletype will produce LC to indicate that loading is complete.

c. Load $XX002_8$ into Register P to obtain a memory map.

d. Depress the START pushbutton. A memory map, having the format shown in Table 4-2, will be produced on the teletype.

Punch the FRTN System Tape

a. Depress the MSTR CLEAR pushbutton.

b. Load 000001_8 into Register P.

c. Insert the PAL-AP tape into the appropriate input device.

d. Depress the START pushbutton. The tape will be loaded into memory.

e. Load $XX100_8$ into Register P.

f. Load 40_8 into Register A. If a teletype is being used, modify the Register-A load by setting bit 1 of the register. Depress the ON switch on the ASR-33 teletype punch or set the MODE switch to KT on the ASR-35 teletype unit.

g. Depress the START pushbutton. The program will halt.

h. Load into Register A location HIGH obtained from the memory map.

i. Depress the START pushbutton. The self-loading FRTN system tape will be punched out by the appropriate output device. To test your system tape, perform the compiling procedure described in Section V.

GENERATE A SELF-LOADING UPDATE SYSTEM TAPE

The following paper tapes from your library are required to generate a self-loading update system program.

a. Some version of SLDR or LDR as determined from Table 4-1

b. PAL-AP (self-loading form)

c. SSUP

d. SSUP-IOS

e. SSUP-RDS

f. I/O Library.

When you have determined which loader is required, proceed as follows.

Load the SSUP Object Tape

a. Perform the procedure for loading the self-loading loader as described earlier in this section.

b. Depress the MSTR CLEAR pushbutton.

c. Load $XX000_8$, the starting location of the loader, into Register P (XX is the highest sector of memory).

d. Insert the SSUP object tape into the appropriate input device.

e. Depress the START pushbutton. The tape will be read into memory starting at location 400_8. When loading is complete, the teletype will produce MR to indicate that additional routines must be loaded.

Load the IOS and the IOS Calls

You must specify which I/O devices you wish to use when updating a symbolic source program. Only one device may be specified for each I/O form as indicated in Table 4-3. After you have determined which devices you wish to use, you must isolate the I/O routines required from the I/O Library tape by loading them individually using the appropriate input device. The procedure is as follows.

a. Insert the SSUP-IOS tape into the appropriate input device.

b. Depress the START pushbutton. When loading is complete, the teletype will produce MR.

c. Insert the required I/O routines into the appropriate input device.

d. Depress the START pushbutton. When loading is complete, the teletype will produce MR.

e. Insert the SSUP-RDS tape into the appropriate input device.

f. Depress the START pushbutton. When loading is complete, the teletype will produce LC.

g. Load $XX002_8$ into Register P to obtain a memory map.

h. Depress the START pushbutton. A memory map will be produced on the teletype having the format shown in Table 4-2.

Punch the SSUP System Tape

To punch out the SSUP system tape, perform the procedure for punching a DAP System Tape as described earlier in this section. The SSUP system tape can be tested by performing an update as described in Section V.

Table 4-3.
I/O Selection for SSUP

I/O Form	I/O Device Desired*	I/O Routines Required
Source Program to be Updated	Magnetic Tape Unit High-Speed Paper Tape Reader	IMA, C6T08, M$UNIT I$PA
Source Program Containing Corrections	ASR Paper Tape Reader High-Speed Paper Tape Reader Card Reader	I$AA I$PA I$CA
Updated Source Program	Magnetic Tape Unit High-Speed Paper Tape Punch	OMA, C8T06, M$UNIT O$PA
Listing of the Corrections	ASR Paper Tape Punch ASR Listing	O$AA O$LL

*The update program requires a high-speed paper tape reader and punch or two magnetic tape units.

SECTION V
OPERATING PROCEDURES

This section presents detailed procedures for loading, executing, assembling and compiling programs, and generating self-loading paper tapes. The assembly, compilation, and updating procedures describe how to use the self-loading system tapes that you generated in accordance with the instructions given in Section IV. Some of the procedures that you performed for specific programs in Section IV have been generalized to cover all programs on paper tape.

LOAD

The following paragraphs contain procedures for loading both self-loading and object programs on paper tape.

Load Self-Loading Programs

a. Depress the MSTR CLEAR pushbutton.

b. Load 000001_8 into Register P.

c. Insert the self-loading tape into the appropriate input device.

d. Depress the START pushbutton. (When loading with an ASR-33 teletype unit, the manual START switch on the device must be activated. When loading with an ASR-35 teletype unit, the MODE switch must be set to KT.) (Observe Register X which should indicate counting if the tape is being loaded correctly.) The program will be loaded into the locations it occupied when it was punched out. Loading any self-loading tape destroys the contents of locations 00020_8 through 00057_8 inclusive.

Load Object Programs

a. Perform the procedure for self-loading programs described above using the appropriate self-loading loader program.

b. Load $XX000_8$, the starting location of the loader, into Register P (XX is the highest sector of memory). If the object program is relocatable (REL), perform steps c. and d. If the object program is absolute (ABS), skip steps c. and d. and continue at step e.)

c. Load Register A with the octal address of the location at which loading is to begin. Otherwise a starting location of 1000_8 will be assumed. To force a starting location of 00000_8, load 100000_8 into Register A. (Locations 1 through 17 octal in memory are write protected and nothing will be loaded into this area.)

d.   Load Register B with the octal address of the location at which the inter-sector indirect address word table is to begin.   Otherwise, a starting location of 100_8 will be assumed.

NOTE

The inter-sector indirect word table must be located in sector 0 for DDP-516 computers without the Memory Lockout Option.   The table may be in any sector for computers with the option.   The sector may be changed at load time by the pseudo-operation SETB.

e.   Insert the object program into the appropriate input device.

f.   Depress the START pushbutton.   The object program will be loaded into memory until a halt occurs and a loader message is produced on the teletype.   Refer to Table 5-1 for the significance of the loader messages.

Table 5-1.
Loader Messages

Message	Meaning	Action Required
LC	Loading complete	Depress the START pushbutton to begin program execution
MR	More subroutines required	Insert the required subroutine tapes into the appropriate input device and depress the START pushbutton to continue loading
CK	Checksum error in the last block read	Depress the START pushbutton to ignore the block and continue loading. A valid load is not assured
BL	Block too large or improperly formatted	Depress the START pushbutton to ignore the block and continue loading. A valid load is not assured
MO	Memory overflow due to program attempting to overwrite the loader	Depress the START pushbutton to obtain a memory map. Refer to Table 4-2 for the memory map format. No recovery is possible from this error.

The loader cannot detect a program overlaying the indirect word tables or the tables overlaying the program.   You must obtain a memory map to determine whether overlap exists. The loader will detect base sector overflow and produce an MO message.

Options Following a Loader Halt

When a loader halt occurs and a message is produced, there are several options you can perform other than those specified in Table 5-1.

The options allow you to:

a.   Reload the object tape by repeating the procedure for load object programs described above.

b.   Recover from a missing end-of-tape block.   The procedure is as follows.

(1)   Momentarily set the MS/SI/RUN switch to SI to stop the computer.

(2)   Load $XX001_8$ into Register P.

(3)   Depress the START pushbutton.

c. Print a memory map. The procedure is as follows.

    (1) Load $XX002_8$ into Register P.

    (2) Depress the START pushbutton. A memory map will be produced on the teletype. See Table 4-2 for the format of the memory map.

d. Set a program break. The procedure is as follows.

    (1) Load $XX003_8$ into Register P.

    (2) Load Register A with the address of the location at which loading is to continue. If Register A is cleared, the origin for loading remains unchanged.

    (3) Depress the START pushbutton and loading will continue.

e. Force-load a subprogram. The procedure is as follows.

    (1) Load $XX004_8$ into Register P.

    (2) Insert a tape into the appropriate input device if required.

    (3) Depress the START pushbutton. The tape will be loaded into memory.

f. Begin executing the object program. The procedure is as follows.

    (1) Load $XX005_8$ into Register P.

    (2) Depress the START pushbutton and the program will be executed.

ASSEMBLE DAP-16 SOURCE PROGRAMS

This procedure enables you to perform an assembly by using your self-loading DAP-16 system tape. To perform the assembly, proceed as follows.

a. Depress the MSTR CLEAR pushbutton.

b. Load 000001_8 into Register P.

c. Insert the DAP-16 system tape into the appropriate input device.

d. Depress the START pushbutton and the system tape will be loaded into memory. Monitor Register X for counting.

e. Select a bit pattern from Table 5-2 to designate a one- or two-pass assembly and to designate the I/O devices to be used during the assembly.

f. Load the selected bit pattern into Register A.

g. Load 400_8 into Register P. There are five standard starting options available for the DAP-16 Assembler.

    (1) 400_8 - Start assembly

    (2) 401_8 - Continue assembly

    (3) 402_8 - Start subroutine tape assembly

    (4) 403_8 - Terminate assembly

    (5) 404_8 - Restart pass two to produce additional object tapes.

h. Insert a DAP-16 source tape into the appropriate input device and turn on your input/output equipment.

NOTE

If you are using the high-speed paper tape equipment or an ASR-35 to assemble, perform step i to complete the procedure. If you are using the ASR-33 to assemble, skip step i and perform steps j through n.

i. Depress the START pushbutton and the assembly will be executed.

j. Depress the START pushbutton. A portion of the source tape will be read and a halt will occur.

k. Depress the ON pushbutton on the ASR-33 paper tape punch.

l. Depress the START pushbutton. A length of object tape will be punched and a halt will occur.

m. Depress the OFF pushbutton on the ASR-33 paper tape punch.

n. Repeat steps j through m until the complete source tape has been read and a complete object tape has been punched. (If after a punch sequence the source tape is not read as in step j, repeat steps k through n. This occurs when there are two blocks to be punched.)

After the assembly is completed, you will have an object tape and a listing of your program if these were requested. When operating in the two-pass assembly mode, the output is generated during the second pass which is accomplished by repeating steps h. and i. after the source tape has been read the first time. If the assembly was successfully completed, Register A will contain all ONEs when the computer halts. If a MOR pseudo-operation was encountered, Register A will contain all ZEROs.

The DAP-16 assembler indicates coding errors by typing or printing error flags in the left-hand margin of the listing. (See Table 9-1.) Such errors do not interfere with the assembly process. Undefined symbols are automatically defined by the Assembler and are listed at the end of the last-pass.

Table 5-2
Assembly and I/O Device Selection

Set Bit	Assembly or I/O Device Selected
1	Two-pass assembly. If bit 1 is not set, a one pass assembly is executed.
	Input Device
2	Teletype Unit
3	High-Speed Paper Tape Reader
4	Card Reader
5	Magnetic Tape Unit
6	Teletype Unit with halts for manual inputs
	Output Device
7	Teletype Unit
8	High-Speed Paper Tape Punch
9	Card Punch
10	Magnetic Tape Unit
11	No Object Output
	Listing Device
12	Teletype Unit
13	High-Speed Paper Tape Punch
14	Magnetic Tape Unit
15	Line Printer
16	No Listing

NOTE

If any of the five-bit I/O groups are all ZEROs, a standard device, depending on the configuration, will be selected.

COMPILE FORTRAN IV SOURCE PROGRAMS

This procedure enables you to compile FORTRAN IV source tapes by using your self-loading FORTRAN IV system tape. The procedure is as follows.

    a.   Depress the MSTR CLEAR pushbutton.

    b.   Load 000001_8 into Register P.

    c.   Insert the FORTRAN IV system tape into the appropriate input device.

    d.   Depress the START pushbutton and the system tape will be loaded into memory.

    e.   Select a bit pattern from Table 5-3 to designate the I/O devices to be used during the compilation.

    f.   Load the selected bit pattern into Register A.

    g.   Load 1000_8 into Register P.

    h.   Set the SENSE switches to desired positions. (See Table 5-4.)

    i.   Insert a FORTRAN IV source tape into the appropriate input device and turn on your input/output equipment.

    j.   Depress the START pushbutton and the compilation will be executed.

When the compilation has been completed, the teletype will produce "End of Job". The selected devices will generate a listing and an object tape provided that these outputs have not been suppressed. The FORTRAN IV compiler indicates coding errors, typing or printing errors by printing error flags on the listing following the FORTRAN statement containing the error. (See Table 9-2.) Some errors are recoverable and do not interfere with the compilation. Others, however, halt the computer and must be corrected before continuing.

Table 5-3.
I/O Device Selection

Bit Selection	I/O Device Selected
Contents of bits 8-10 1 2 3	Input Device Teletype Unit Card Reader High-Speed Paper Tape Reader
Contents of bits 11-13 0 1 2	Listing Device Suppress Listing Teletype Unit Line Printer
Contents of bits 14-16 0 1 2	Output Device Suppress Output High-Speed Paper Tape Punch Teletype Unit

<div align="center">

Table 5-4.
Sense Switch Settings

</div>

Sense Switch	Significance
1	Expand the symbolic listing to include side-by-side octal information
2	Inhibit symbolic listing (not valid if Sense Switch 1 is set)
3	Halt before inputting next record. During the pause (A) will display the current status of the I/O keyboard. The status may be changed before resetting Sense Switch 3 and depressing the START pushbutton
4	Cause trace-coupling information to be included in the object coding being generated regardless of any trace statements in the program. (This is an operator override.)

DEBUG PROGRAMS

Your software package includes a utility program called COP (Check-Out Program) which is designed to assist you in debugging programs. The COP program is relocatable and therefore can be loaded into memory so as not to interfere with the program being debugged. The following procedures describe loading and executing COP.

Loading COP

This procedure assumes that the object program to be debugged has been loaded into memory using the procedure for loading object programs which is described earlier in this section.

    a.   Load $XX002_8$ into Register P to obtain a memory map.

    b.   Depress the START pushbutton. A memory map will be produced on the teletype. (See Table 4-2 for map format.)

    c.   Load the COP object program into memory by using the procedure for loading object programs which is described earlier in this section. (COP may utilize locations in sector zero for cross-sector linkage. Refer to the memory map before selecting the starting locations to be loaded into Registers A and B as part of the loading process.)

Execute COP

When COP has been loaded into memory, there are a number of operations that you can perform with the teletype. These operations are explained in Table 5-5 where the first column designates the operation, the second column specifies the keys used on the teletype, and the third column explains the operation performed.

Table 5-5.
COP Operations

Operation	Teletype Keys Activated*	Explanation
A	A RETURN X, Y RETURN	Type the octal contents of memory from location X to location Y.
B	B RETURN X, Y RETURN (.)	Insert correction Y into location X. Type Y RETURN to insert second and successive corrections. A slash (/) will be printed out when the memory modification has been completed.
C	C RETURN X, Y RETURN	Enter breakpoint at location X and start program at location Y. Location 511 of sector zero is used to RETURN from a breakpoint.
D	D RETURN	RETURN to the breakpoint and continue the program being debugged. Clear breakpoint, only one breakpoint active at a time.
E	E RETURN X, Y RETURN	Clear the contents of memory from location X to location Y.
F	F RETURN X, Y RETURN Z RETURN	Search memory between locations X and Y for address Z.
G1	G1, A, B, X, C RETURN X, Y RETURN	Start at location X and print the contents of the specified registers every time control passes through location Y.
G2	G2, A, B, X, C RETURN X, Y RETURN N RETURN	Start at location X and print the contents of the specified registers every Nth time control passes through location Y.
G3	G3, A, B, X, C RETURN X, Y RETURN I RETURN N RETURN	Start at location X and print the contents of the specified registers the first I times of every N times control passes through location Y.
G4	G4, A, B, X, C, RETURN X, Y RETURN N RETURN	Start at location X and print the contents of the specified registers for the first N times control passes through location Y.
G5	G5, A, B, X, C, RETURN X, Y RETURN	Start at location X and print the contents of the specified registers every time control passes through location Y.
G6	G6, A, B, X, C RETURN X, Y RETURN A or B or M space > or = or < Space D RETURN Z RETURN (Z is entered only if a memory location M is to be tested.)	Start at location X and print the contents of the specified registers every time control passes through Y if the specified condition is true.
G7	G7, A, B, S, C, RETURN X RETURN	Start at location X and print the contents of the specified registers (trace).

*X, Y, and Z denote octal numbers. D, I, and N denote decimal numbers. In the G operations, A, B, X, and C denote registers. Any combination of these registers may be specified. The checkpoint and its contents are printed whether or not registers are specified.

UPDATE SOURCE PROGRAMS

Your software package contains a symbolic source update program (SSUP) which will enable you to delete, correct, preserve or add records to a symbolic source program. The following procedures describe loading and executing SSUP.

Load SSUP

To load program SSUP, proceed as follows.

a. Depress the MSTR CLEAR pushbutton.

b. Load 000001_8 into Register P.

c. Insert the SSUP system tape into the appropriate input device.

d. Depress the START pushbutton and the system tape will be loaded into memory. Monitor Register X for counting.

e. Insert the source tape to be updated into the appropriate input device.

f. Set SENSE SWITCH 1 to the appropriate position. The up position (set) selects the FORTRAN IV updating option; the down position (reset) selects the DAP-16 updating option.

Execute SSUP

When SSUP has been loaded into memory, there are a number of operations you can perform using the teletype which allows you to modify the old source tape and punch a new source tape which reflects the modifications. These operations are explained in Table 5-6 where the first column designates the operation command, the second column specifies the keys used on the teletype, and the third column explains the operation performed. Refer to the SSUP program listing (3C Doc. No. 180767000) for a complete description of the rules for using SSUP.

Table 5-6.
SSUP Operations

Operation Command	Teletype Keys Activated ▽1	Explanation
List	$LIST (*F*L1*, *L2)	Make a listing of the program including corrections as specified by L1 and L2.
No List	$NLST	Do not make a listing
Begin	$BEGN (*F*L1)	Position the new source tape to the file or record specified by L1.
Halt	$HALT	Halt SSUP and print a breakpoint halt on the listing.

▽1 An asterisk (*) indicates permissible spaces and the entries within the parenthesis indicate optional operations dependent upon the command.

Table 5-6. (Cont)
SSUP Operations

Operation Command	Teletype Keys Activated ▽1	Explanation
Reset	$RSET (*F*L1)	Set the internal file or record counter for the tape being updated to the value specified by L1.
No Copy	$NCPY	Do not write onto the new source tape.
Copy	$COPY (*F*L1*, *L2)	Copy onto the new source tape that information specified by L1 and L2.
Done	$DONE	Terminate the updating process.
Insert	$NSRT (*F*L1*, *L2)	Insert new information onto the new source tape which does not appear on the old source tape.
Omit	$OMIT (*F*L1*, *L2)	Delete information that appears on the old source tape by not copying it onto the new source tape.
End of File	$WEOF	Write an end of file mark on the output magnetic tape.

▽1   An asterisk (*) indicates permissible spaces and the entries within the parenthesis indicate optional operations dependent upon the command.

DUMP THE CONTENTS OF MEMORY

Your software package contains a memory dump program which is appropriately called DUMP. The program enables you to print selected areas of memory on the teletype in any one of three formats. The procedure for using DUMP is as follows.

a.   Depress the MSTR CLEAR pushbutton.

b.   Load $XX002_8$ into Register P and depress the START pushbutton to obtain a memory map.

c.   Load the DUMP program into memory by using the procedure for loading object programs that was described earlier in this section.

d.   Load into Register P the address at which the DUMP program begins. (This address is determined from the memory map and is loaded into Register A during the loading procedure performed in step c.)

e.   Depress the START pushbutton. The program performs a carriage return, a line feed and awaits instructions via the teletype.

f.   Select the appropriate sense switch settings to obtain the desired dump format. The selections are as follows.

Sense Switch Setting	Format Option
1 Set	Octal printout with eight columns per row.
2 Set	Mnemonic printout with four columns per row.
1 and 2 Reset	Mnemonic printout with one column per row.

g. Specify the limits of the dump by typing in X CARRIAGE RETURN and Y CARRIAGE RETURN where X and Y are octal numbers specifying the lower and upper limits of the dump, respectively. The contents of the specified area of memory will then be printed by the teletype. If an error is made in the entry format, a slash (/) will be typed and the arguments must be entered correctly.

PUNCH SELF-LOADING OBJECT TAPES

A program is provided in your software package which enables you to punch self-loading object tapes of any segment of memory. This punch and load program (PAL-AP) is supplied in self-loading form. The following procedures describe the method for loading the program and executing the program once it is loaded.

Load the Self-Loading PAL-AP

a. Depress the MSTR CLEAR pushbutton.

b. Load 000001_8 into Register P.

c. Depress the START pushbutton. The tape will be loaded into the highest 1200_8 locations of memory. (The program overlays the relocating loader if it is currently in the high sectors of memory and locations 00020_8 through 00057_8 are destroyed.)

Execute PAL-AP

a. Depress the MSTR CLEAR pushbutton.

b. Load ZZ100 into Register P. (ZZ is the sector into which PAL-AP was loaded.)

c. Load into Register A the starting address of the data to be punched out. (If the teletype is being used, set the high-order bit of Register A and turn on the punch unit on the teletype.)

d. Depress the START pushbutton. The program will halt at location $ZZ117_8$.

e. Load into Register A the ending address of the data to be punched out.

f. Depress the START pushbutton. A self-loading object tape will be punched out.

SECTION VI
UTILITY LIBRARY

This section contains short descriptions of the routines in the DDP-516 Utility Library. The purpose, storage requirements, and usage are given for each routine. Errors that may occur and other routines called are also given where applicable. In some cases more detailed information contained either in the program listing or other documents is referenced.

DAP ASSEMBLER (DAP 16)

Purpose

The DAP assembler is designed to convert DAP source language programs to object form for execution on the DDP-116, 416, or 516. DAP-16 is a one- or two-pass assembler. Additional passes may be made to produce extra copies of the program in object form.

Storage Requirements

DAP-16 requires a minimum of 4K core storage using IOS-16A (preselected I/O), or 8K using IOS-16B (selectable I/O).

Usage

Refer to the DAP-16 Reference Manual, 3C Doc. No. 130071629, to the program listing, 3C Doc. No. 180275000, and to section V of the manual for complete details on use and operation.

FORTRAN IV COMPILER (FRTN)

Purpose

The FORTRAN IV Compiler is designed to compile ASA standard FORTRAN IV programs in one pass and generate optimum object code for execution on the DDP-116, or 516.

Storage Requirements

FRTN requires a minimum of 8K storage.

Usage

Refer to FORTRAN IV Manual, 3C Doc. No. 130071479 and section V of this manual for complete details on usage.

PUNCH AND LOAD (PAL-AP)

Purpose

PAL-AP is used to punch self-loading object tapes for execution on the DDP-516. PAL-AP consists of a punch section and a load section. The load section is of the bootstrap variety. PAL-AP will first punch out its own loader section in 8-8 format followed by 12 in. of leader. The desired program is then punched in PAL format, which is recognized by the loader. The self-loading ability of a PAL program may thus be seen. The loader on the front of the tape will load itself and then the PAL-format program.

Storage Requirements

PAL-AP requires less than one sector of memory.

Usage

Refer to Section V of this manual for complete procedures for using PAL-AP and to 3C Doc. No. 180311000 for a program listing.

EXPANDED LOADER (LDR-A)

Purpose

LDR-A loads absolute and relocatable main programs and subprograms produced by the DAP-16 Assembler (in a one or two-pass mode) or the FORTRAN IV Compiler into the DDP-516 memory. LDR-A loads paper tape through the ASR teletypewriter.

Storage Requirements

LDR-A requires approximately 3 1/2 sectors of memory.

Usage

Refer to program listing, 3C Doc. No. 180335000, and to Section V of this manual for complete details on usage.

EXPANDED LOADER (LDR-P)

Purpose

LDR-P loads absolute and relocatable main programs and subprograms produced by the DAP-16 Assembler (in a one or two-pass mode) or the FORTRAN IV Compiler into the DDP-516 memory. LDR-P loads tape through the high-speed paper tape reader.

Storage Requirements

LDR-P requires approximately 3 1/2 sectors of memory.

Usage

Refer to program listing, 3C Doc. No. 180336000, and to Section V of this manual for complete details on usage.

STANDARD LOADER (SLDR-A)

Purpose

SLDR-A loads absolute and relocatable main programs and subprograms produced by the DAP-16 Assembler (two-pass mode only) or the FORTRAN IV Compiler into the DDP-516 memory. SLDR-A loads paper tape through the ASR tape reader. SLDR-A may not be used with Extended Addressing and Memory Lockout options.

Storage Requirements

SLDR-A requires more than two sectors of memory.

Usage

Refer to program listing, 3C Doc. No. 180341000, and to Section V of this manual for complete details on usage.

STANDARD LOADER (SLDR-P)

Purpose

SLDR-P loads absolute and relocatable main programs and subprograms produced by the DAP-16 Assembler (two-pass mode only) or the FORTRAN IV Compiler into the DDP-516 memory. SLDR-P loads paper tape through the high-speed paper tape reader. SLDR-P may not be used with Extended Addressing and Memory Lockout options.

Storage Requirements

SLDR-P requires more than two sectors of memory

Usage

Refer to program listing, 3C Doc. No. 180342000, and to Section V of this manual for complete details on usage.

MEMORY DUMP - OUTPUT ON ASR-33 (DUMP)

Purpose

Print out selected areas of core on the ASR-33 typewriter with three different format options as follows:

a. If sense switch 1 is set, memory is printed octally between specified limits with eight columns per row.

b. If sense switch 2 is set, memory is printed in mnemonic format between specified limits with four columns per row.

c. If sense switches 1 and 2 are reset, memory is printed in mnemonic format between specified limits with one column per row.

Storage Requirements

DUMP requires one sector of memory.

Usage

Refer to the program listing, 3C Doc. No. 188806000, and to Section V of this manual for complete operating procedures.

Errors

Arguments entered incorrectly. A slash will be typed and the user must re-enter the arguments correctly.

CHECK-OUT PROGRAM (COP)

Purpose

COP aids in check-out and debugging of DDP-516 programs by permitting the user to print out selected areas of core (trace option), delete or insert single words or blocks of words, enter breakpoints, initiate jumps or jumps and halts, and search all or any part of memory for references to specific addresses.

Storage Requirements

COP requires 628_{10}(1164_8) locations (Rev. A).

Usage

Refer to program listing, 3C Doc. No. 188807000, and to Section V of this manual for complete details on options available and their use.

Errors

Incorrect format for calling an option. A slash will be typed and the user must re-enter the call for the option correctly.

SYMBOLIC SOURCE UPDATE PROGRAM (SSUP)

Purpose

SSUP reads symbolic records that are less than 80 characters long and permits a user to delete, correct, preserve, or add records to the symbolic source to create a new revised symbolic source. SSUP is compatible with any peripheral device that can be controlled by IOS, and will operate on any DDP-516 having either the high-speed paper-tape reader and punch option or two magnetic tape units.

Storage Requirements

SSUP requires $1055_{10}(2037_8)$ locations (Rev. B).

Usage

Refer to program listing, 3C Doc. No. 180767000, and to Section V of this manual for a complete description of the use of commands, the procedure for making corrections, and error messages.

Errors

For complete details of invalid commands or incorrect limits, refer to the program listing.

Other Routines Called

SSUP-IOS

SYMBOLIC SOURCE UPDATE, INPUT/OUTPUT SUPERVISOR (SSUP-IOS)

Purpose

SSUP-IOS selects the proper input/output device required by SSUP, and causes it to input to or output from buffer areas provided by SSUP. SSUP never calls an input/output routine directly, but rather calls on SSUP-IOS when it is required to input the updating master or old master, or output the new master, the print master, or comments. SSUP also calls on SSUP-IOS for initialization of a heading line and when an end-of-job or an end-of-file condition is encountered.

Storage Requirements

SSUP-IOS requires $161_{10} (241_8)$ locations (Rev. A).

Usage

Refer to the program listing, 3C Doc. No. 180000000, for complete details on usage and method.

Other Routines Called

OLH, IAA, OME,   IMA, C$T06, O$LA,

OAH, IPA, OPS,   IPA, OMA, OLL,

ICA, OAL, O$AS   C$T08, O$PA

SYMBOLIC SOURCE UPDATE - REVISED DUMMY SELECTION (SSUP-RDS)

Purpose

SSUP-RDS satisfies IOS calls for input/output routines that have not been loaded. This subroutine is loaded after all input/output routines. Its prime use is in the preparation of an SSUP systems tape. The subroutine is entered at execution time only when a non-selected I/O subroutine is called by IOS. It automatically steps through the calling sequence, and then returns directly to the IOS to call on the next I/O subroutine on the list.

Storage Requirements

SSUP-RDS requires $25_{10} (31_8)$ locations (Rev. A).

Method

Refer to program listing, 3C Doc. No. 180304000, for details on the method used.

CHAIN OR SEGMENT (CHAIN)

Purpose

CHAIN is used to read in the next segment of a "chained" program. The chain segment is assumed to be written in load-mode on paper tape, to be read by the paper tape reader, and to start at location 1000_8. The size of memory is determined and the modified load mode program is transferred to the end of memory. The segment from the paper tape reader is then loaded in and control is transferred to 1000_8.

Storage Requirements

CHAIN requires $67_{10} (102_8)$ locations (Rev. A).

Usage

The calling sequence for this routine is as follows.

    CALL   CHAIN

INDIRECT TO DIRECT ADDRESS (ARG$)

Purpose

ARG$ provides a software representation of indirect addressing by converting the indirect address of an argument to its corresponding direct address and storing it in the index register.

Storage Requirements

ARG$ requires $15_{10}(17_8)$ locations (Rev. A).

Usage

The calling sequence for this routine is as follows.

    CALL   ARG$
    DAC*   (argument address)

ARGUMENT TRANSFER (F$AT)

Purpose

F$AT transfers a variable number of arguments from the calling routine to the called routine. The number of arguments are specified by the first pseudo-operation following the call. The second pseudo-operation specifies the beginning location of the block into which the arguments are to be placed.

Storage Requirements

F$AT requires $49_{10}(61_8)$ locations (Rev. A).

Usage

The calling sequence for this routine is as follows.

                        (entry to routine)
    CALL   F$AT
    DEC   N   (number of arguments to pass)
    DAC   A   (pointer to storage block for arguments)

ERROR ENTRY OR HALT (FER, FHT)

Purpose

F$ER causes a mnemonic error indicator to be printed on the ASR-33 when an object time error is encountered in a specific routine. F$HT causes the computer to stop and print PA if a PAUSE statement has been encountered, or to print ST if a STOP statement has been encountered.

Storage Requirements

F$ER and F$HT require $36_{10}(44_8)$ locations (Rev. A).

Usage

The calling sequence for F$ER is

```
CALL    F$ER
DAC     ARG1
```

where ARG1 is the address of the indicator to be printed out. The routine will print out the error indicator and then halt if SENSE SWITCH 3 is set. Depressing the START push-button after the halt will cause the program to continue. If SENSE SWITCH 3 is not set, F$ER will exit with no printout or halt taking place.

The calling sequence for F$HT is as follows.

```
CALL    F$HT
DAC  =  '151724
```

This calling sequence is normally generated when the STOP statement has been encountered in the compilation process. The literal specified is the ASCII equivalent of ST which is the object time indicator specifying that a stop has been encountered.

OVERFLOW - SET ERROR FLAG (OVERFL)

Purpose

OVERFL is used to check error conditions. The error flag is turned off. Indicator J is set to 1 if the error flag was on, or set to 2 if the error flag was off. The error flag is located in subroutine F$ER and is set anytime an error occurs in an arithmetic function calculation or in an input/output conversion.

Storage Requirements

OVERFL requires $14_{10}(16_8)$ locations (Rev. A).

Usage

The calling sequence for OVERFL is as follows.

DAP		FORTRAN
CALL	OVERFL	CALL OVERFL (J)
DAC	J	
(return)		

PSEUDO SENSE LIGHTS (SLITE, SLITET, SSWTCH)

Purpose

SLITE (I)   -   To set sense light I; or if I=O, to reset all sense lights.

SLITET (I, J) -   To test sense light I and turn it off.

               If sense light I was set, indicator J is set to 1.

               If sense light I was reset, indicator J is set to 2.

SSWTCH (I, J) -   To test sense switch I and to set indicator J to 1 if I is

               set or to set indicator J to 2 if I is reset.

Storage Requirements

Routines SLITE, SLITET, and SSWTCH require $73_{10}(111_8)$ locations (Rev. A).

Usage

The calling sequence for SLITE is as follows.

DAP		FORTRAN
CALL	SLITE	CALL SLITE (I)
DAC	I (address of variable containing sense light number)	

The calling sequence for SLITET is as follows.

DAP		FORTRAN
CALL	SLITET	CALL SLITET (I, J)
DAC	I (address of variable containing sense light number)	
DAC	J (address for storing indicator)	

The calling sequence for SSWTCH is as follows.

DAP		FORTRAN
CALL	SSWTCH	CALL SSWTCH (I, J)
DAC	I (address of variable containing sense switch to be interrogated)	
DAC	J (address for storing indicator)	

Other Routines Called

ARG$ and L$33

LOGICAL COMPLEMENT (N$33)

Purpose

N$33 is used to obtain the complement of a logical argument by changing the value of the least significant bit in Register A from 0 to 1 or from 1 to 0.

Storage Requirements

N$33 requires $4_{10}(4_8)$ locations (Rev. A).

Usage

The calling sequence for N$33 is as follows.

    CALL   N$33

LOGICAL OR FROM MEMORY (L$33)

Purpose

L$33 is used to form an inclusive OR from memory with the argument in Register A. The argument in Register A is first exclusive ORed with the argument from memory. The result is ANDed with the argument from memory and this result is then exclusive ORed with the argument from memory, thus forming the inclusive OR.

Storage Requirements

L$33 requires $12_{10}(14_8)$ locations (Rev. A).

Usage

The calling sequence for L$33 is as follows.

    CALL   L$33
    DAC   ARG1

SECTION VII
MATHEMATICAL LIBRARY

All mathematical routines in the DDP-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. 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, as follows.

The integer accumulator is Register A.

The single-precision, or real, accumulator includes the 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}.

The double-precision accumulator is memory locations 75_8, 76_8 and 77_8, with the sign in bit position 1 of 75_8, the exponent in bit positions 2-9 of 75_8, and the fraction in bit positions 10-16 of 75_8, 1-16 of 76_8, and 1-16 of 77_8.

The complex accumulator is memory locations 74_8 through 77_8. The sign of the real part is in bit position 1 of 74_8, the exponent is in bit positions 2-9 of 74_8, and the fraction is in bit positions 10-16 of 74_8 and bit positions 1-16 of 75_8. The imaginary part of the complex number occupies words 76_8 and 77_8 in the same manner.

When you are not using FORTRAN IV and you wish to use an integer or single-precision subroutine that requires more than one argument, you can place the first argument in Register A, or Registers A and B, via 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, you can load the accumulators via the L\$66 and L\$55 subroutines, as follows.

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 follows.

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

The symbolic expressions given are to be 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 (3C Doc. No. 180092000). The FORTRAN IV Manual (3C 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	Timing (µsec)	Storage (Decimal)	Other Routines Used
Complex:							
Absolute value.	CABS	Call CABS, DAC Arg 1	R=CABS(C)	None	2926.1	27	FAT,L22,M$22,H$22,A$22,SQRT
Add	A$55	Call A$55, DAC Arg 2	C=C+C	None	819.8	35	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	503.0	20	FAT,H55,L$22,H$22,L$55
Conjugate.	CONJ	Call CONJ, DAC Arg 1	C=CONJG(C)	None	307.2	30	FAT,L22,H$22,N$22,L$55
Convert imaginary part to real.	AIMAG	Call AIMAG, DAC Arg 1	R=AIMAG(C)	None	79.9	9	L$55,L$22
Cosine	CCOS	Call CCOS, DAC Arg 1	C=CCOS(C)	None	18067.2	14	FAT,L55,A$55,H$55,CSIN
Divide	D$55	Call D$55, DAC Arg 2	C=C/C	None	6047.0	87	FAT,H55,L$22,M$22,H$22,A$22,DS22,S$22,L$55
Divide by single-precision argument	D$52	Call D$52, DAC Arg 2	C=C/R	None	2379.8	30	FAT,H55,L$22,DS22,H$22,L$55
Exponential, base e	CEXP	Call CEXP, DAC Arg 1	C=CEXP(C)	None	14458.6	42	FAT,EXP,H22,COS,M$22,SIN,L$55
Load	L$55	Call L$55, DAC Arg 1	C=C	None	39.4	13	ARG$
Logarithm, base e	CLOG	Call CLOG, DAC Arg 1	C=CLOG(C)	None	9478.1	52	F$AT,LS22,M$22,H$22,A$22,ALOG,ATAN2,L$55
Multiply	M$55	Call M$55, DAC Arg 2	C=C*C	None	2731.2	64	FAT,H55,L$22,M$22,S$22,A$22,L$55
Multiply by single-precision argument.	M$52	Call M$52, DAC Arg 2	C=C*R	None	1212.5	32	FAT,H55,L$22,M$22,H$22,L$55
Negate	N$55	Call N$55, DAC Arg 1	C=-C	None	345.8	25	H$55,L$22,N$22,H$22,L$55
Raise to integer power	E$51	Call E$51, DAC Arg 1	C=C**I	None	3094.1	41	FAT,H55,IABS,L$55,M$55,D$55
Sine	CSIN	Call CSIN, DAC Arg 1	C=CSIN(C)	None	17047.7	59	FAT,EXP,H22,L$22,DS22,A$22,SIN,M$22,L$55,S$22,COS
Square root	CSQRT	Call CSQRT, DAC Arg 1	C=CSQRT(C)	None	7189.4	68	FAT,ABS,H22,CABS,A$22,M$22,SQRT,L$22,SIGN,DS22,L$55
Store (hold)	H$55	Call H$55, DAC Arg 1	C=C	None	39.4	13	ARG$
Subtract	S$55	Call S$55, DAC Arg 2	C=C-C	None	827.5	35	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	506.9	20	FAT,H55,L$22,S$22,H$22,L$55
Double-precision:							
Fixed-point:							
Addition.	DADD	Call DADD, DAC Arg 2	D=D+D	Overflow; return to call plus 2	31.7	27	None
Arctangent	DATNX1	Call DATNX1, DAC Arg 1	D=DATNX1(D)	None	4447.7	147	TWOS,DDIV,DMPY,DADD,RODD
*Arctangent	DATNX2	Call DATNX2, DAC Arg 1	D=DATNX2(D)	None	1304.6	147	TWOS,DDIVH,DMPYH,DADD,RODD
Cosine	DCOSX1	Call DCOSX1, DAC Arg 1	D=DCOSX1(D)	None	4306.6	5	DSINX1
*Cosine	DCOSX2	Call DCOSX2, DAC Arg 1	D=DCOSX2(D)	None	1215.4	5	DSINX2
Divide	DMPY	Call DDIV, DAC Arg 2	D=D/D	Divisor ≤ dividend	1217.3	223	MPY,TWOS,DIV
*Divide	DMPYH	Call DDIVH, DAC Arg 2	D=D/D	Divisor ≤ dividend	205.4	203	TWOS
*Exponential, base e	DEXEX1	Call DEXEX1, DAC Arg 1	D=DEXEX1(D)	None	3915.8	138	DADD,RODD,DMPY,DDIV,DSUB
*Exponential, base e	DEXEX2	Call DEXEX2, DAC Arg 1	D=DEXEX2(D)	None	1009.0	138	DADD,RODD,DMPYH,DDIVH,DSUB
Exponential, base 2	DEX2X1	Call DEX2X1, DAC Arg 1	D=DEX2X1(D)	Argument ≥ 0	3881.4	91	RODD,DMPY,DADD,DDIV,DSUB
*Exponential, base 2	DEX2X2	Call DEX2X2, DAC Arg 1	D=DEX2X2(D)	Argument ≥ 0	924.5	91	RODD,DMPYH,DADD,DDIVH,DSUB
Logarithm, base e	DLGEX1	Call DLGEX1, DAC Arg 1	D=DLGEX1(D)	1/e > argument ≥ e	5158.1	82	DLG2X1,DMPY,DSUB,DADD
*Logarithm, base e	DLGEX2	Call DLGEX2, DAC Arg 1	D=DLGEX2(D)	1/e > argument ≥ e	1496.6	82	DLG2X2,DMPYH,DSUB,DADD
Logarithm, base 2	DLG2X1	Call DLG2X1, DAC Arg 1	D=DLG2X1(D)	Argument < 1/2	4501.4	97	DADD,RODD,DSUB,DDIV,DMPY
*Logarithm, base 2	DLG2X2	Call DLG2X2, DAC Arg 1	D=DLG2X2(D)	Argument < 1/2	1281.6	97	DADD,RODD,DDIVH,DSUB,DMPYH

*Operates with multiply/divide option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Timing (μsec)	Storage (Decimal)	Other Routines Used
Double-precision - Cont							
Fixed-point (Cont)							
Multiply	DMPY	Call DMPY, DAC Arg 2	D=D*D	None (overflow not possible)	577.9	223	MPY,TWOS,DIV
*Multiply	DMPYH	Call DMPYH, DAC Arg 2.	D=D*D	None (overflow not possible)	136.3	203	MPY,TWOS
Round up binary number	RODD	Call RODD, DAC Arg 1	D=RODD(D)	None	6.7	7	None
Sine	DSINX1	Call DSINX1, DAC Arg 1	D=DSINX1(D)	None	4299.8	74	DMPY,DADD
*Sine	DSINX2	Call DSINX2, DAC Arg 1	D=DSINX2(D)	None	1208.6	74	DMPYH,DADD
Square root	DSQRX1	Call DSQRX1, DAC Arg 1	D=DSQRX1(D)	None	4512.0	90	TWOS,DMPY,DADD,DDIV,RODD
*Square root	DSQRX2	Call DSQRX2, DAC Arg 1	D=DSQRX2(D)	None	1034.9	90	TWOS,DMPYH,DADD,DDIVH,RODD
Subtraction	DSUB	Call DSUB, DAC Arg 2	D=D-D	Overflow; return to call plus 2	33.6	28	None
Two's complement	TWOS	Call TWOS, DAC Arg 1	D=TWOS(D)	None	11.5	12	None
Floating-point:							
Absolute value	DABS	Call DABS, DAC Arg 1	D=DABS(D)	None	173.8	11	FAT,L66,N$66
Add	A$66	Call A$66, DAC Arg 2.	D=D+D	Overflow	345.6	553	ARG$,N$66,FER,H66,L$66
Add single-precision argument	A$62	Call A$62, DAC Arg 2.	D=D+R	None	653.8	12	EAT,H66,DBLE,A$66
Add integer to exponent	A$81	Call A$81, DAC Arg 1.	D=A$81(I)	Overflow	28.8	9	F$ER
Arctangent, principle value	DATAN.	Call DATAN, DAC Arg 1	D=DATAN(D)	None	18778.6	134	F$AT,DABS,H$66,C$81,L$66,A$66,D$66,M$66,DSIGN
Arctangent x/y	DATAN2	Call DATAN2, DAC Arg 2	D=DATAN(D,D)	Y=0	23261.8	56	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	10.6	11	None
Convert exponent to integer	C$81	Call C$81, DAC Arg 1	I=C$81(D)	None	12.5	8	None
Convert to integer	C$61	Call C$61, DAC Arg 1	I=C$61(D)	None	268.8	4	C$62,C$21
Convert to single-precision (from pseudo accumulator)	C$62	Call C$62, DAC Arg 1	R=C$62(D)	None	6.7	5	None
Cosine	DCOS	Call DCOS, DAC Arg 1	D=DCOS(D)	None	14385.6	14	FAT,L66,A$66,H$66,DSIN
Divide	A$66	Call D$66, DAC Arg 2.	D=D/D	Overflow	4057.0	553	ARG$,N$66,FER,H66,L$66
Divide by single-precision argument	D$62	Call D$62, DAC Arg 2.	D=D/R	None	4441.9	16	FAT,H66,DBLE,L$66,D$66
Exponential, base e	DEXP.	Call DEXP, DAC Arg 1	D=DEXP(D)	None	17441.3	103	FAT,M66,H$66,C$61,C$16,N$66,A$66,L$66,S$66,D$66,A$81
Load	L$66	Call L$66, DAC Arg 1	D=D	None	35.5	11	ARG$
Logarithm, base e	DLOG.	Call DLOG, DAC Arg 1	D=DLOG(D)	Negative or zero argument.	15522.2	9	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.	14327.0	83	FAT,M66,L$66,F$ER,C$81,C$16,H$66,S$66,Z$80,D$66,A$66
Logarithm, base 10	DLOG10	Call DLOG10, DAC Arg 1	D=DLOG10(D)	Negative or zero argument.	15526.1	10	F$AT,DLOG2,M$66
Maximum value	DMAX1	Call DMAX1, DAC Arg n.	D=DMAX1(D1,D2,Dn)	None	1790.4	35	L$66,H$66,S$66
Minimum value	DMIN1	Call DMIN1, DAC Arg n	D=DMIN1(D1,D2,Dn)	None	1791.4	36	L$66,H$66,S$66
Multiply	A$66	Call M$66, DAC Arg 2	D=D*D	Overflow	1081.0	553	ARG$,N$66,H$66,L$66
Multiply by single-precision argument	M$62	Call M$62, DAC Arg 2	D=D*R	None	1388.2	12	FAT,H66,DBLE,M$66

*Operates with multiply/divide option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Timing (μsec)	Storage (Decimal)	Other Routines Used
Double-precision - (Cont)							
Floating-point - (Cont)							
Negate	N$66	Call N$66, DAC Arg 1	D=D(-D)	None	22.1	21	None
Raise to double-precision power	E$66	Call E$66, DAC Arg 2	D=D**D	None	34239.4	16	FAT,H66,DLOG,M$66,DEXP
Raise to integer power	E$61	Call E$61, DAC Arg 2	D=D**1	None	2113.9	38	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	34545.6	16	FAT,M62,H$66
Remainder	DMOD	Call DMOD, DAC Arg 2	D=DMOD(D,D)	None	6597.1	20	FAT,L66,D$66,H$66,DINT, M$66,S$66,N$66
Sine	DSIN	Call DSIN, DAC Arg 1	D=DSIN(D)	None	13849.0	116	FAT,L66,M$66,H$66,C$61,C$16, N$66,A$66,MOD,S$66
Square root	DSQRT	Call DSQRT, DAC Arg 1	D=DWQRT(D)	None	6108.5	24	FAT,L66,C$62,H$22,SQRT, C$26,H$66,D$66,A$66,A$81
Store (hold)	H$66	Call H$66, DAC Arg 1	D=D	None	35.5	11	ARG$
Subtract	A$66	Call S$66, DAC Arg 2	D=D-D	Overflow	412.8	553	ARG$,N$66,H$66,L$66
Subtract single-precision	S$62	Call S$62, DAC Arg 2	D=D-R	None	745.9	13	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	191.0	21	FAT,L66,N$66
Truncate fractional bits.	DINT	Call DINT, DAC Arg 1	D=DINT(D)	None	819.8	22	L$66,N$66,A$66,S$66
Integer:							
Absolute value	IABS	Call IABS, DAC Arg 1	I=IABS(I)	None	14.4	9	None
Convert to double-precision	C$16	Call C$16, DAC Arg 1	D=C$16(I)	None	275.5	5	C$12,C$26
Convert (FORTRAN-generated) to single-precision	FLOAT	Call FLOAT, DAC Arg 1	R=FLOAT(I)	None	271.7	8	C$12
Convert to single-precision	C$12	Call C$12, DAC Arg	I=C$12(I)	None	256.3	25	A$22,N$22
Divide	D$11	Call D$11, DAC Arg 2	I=I/I	None	218.9	34	ARG$
Maximum single-precision value	MAX0	Call AMAX0, DAC Arg n	R=AMAX0(I1,I2,...In)	None	336.0	34	FLOAT
Maximum value	MAX0	Call MAX0, DAC Arg n	I=MAX0(I1,I2,...In)	None	57.6	34	FLOAT
Multiply	M$11	Call M$11 DAC Arg 2	I=I*I	None	147.8	19	ARG$
Positive difference	IDIM	Call IDIM, DAC Arg 2	I=I1-MIN(I1,I2)	None	34.6	19	None
Raise to integer power	E$11	Call E$11, DAC Arg 2	I=I**I	None	405.1	80	ARG$,M$11,F$ER
Remainder	MOD	Call MOD, DAC Arg 2	I=MOD(I,I)	None	410.9	22	D$11,M$11
Transfer sign of second argument to first.	ISIGN	Call ISIGN, DAC Arg 2	I=ISIGN(I,I)	None	34.6	21	None
Single-precision:							
Fixed-point:							
Arctangent	ATNX1	Call ATNX1, DAC Arg 1	R=ATNX1(R)	None	1057.0	36	MPY,ROND
*Arctangent	ATNX2	Call ATNX2, DAC Arg 2	R=ATNX2(R)	None	146.9	33	ROND
Cosine	COSX1	Call COSX1, DAC Arg 1	R=COSX(R)	None	812.2	5	SINX1
*Cosine	COSX2	Call COSX2, DAC Arg 2	R=COSX2(R)	None	53.8	5	SINX2
Divide	DIV	Call DIV, DAC Arg 2	R=R/R	Dividend ≥ divisor	220.8	80	DIVS
Exponential, base e	EXEX1	Call EXEX1, DAC Arg 1	R=EXEX1(R)	None	838.1	75	MPY,DIV,ROND

*Operates with multiply/divide option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Timing (μsec)	Storage (Decimal)	Other Routines Used
Single-precision - (Cont)							
Fixed-point - (Cont)							
*Exponential, base e	EXEX2	Call EXEX2, DAC Arg 1	R=EXEX2(R)	None	110.4	73	ROND
Exponential, base 2	EX2X1	Call EX2X1, DAC Arg 1	R=EX2X1(R)	Positive or zero argument.	815.0	48	MPY,ROND,DIV
*Exponential, base 2	EX2X2	Call EX2X2, DAC Arg 1	R=EX2X2(R)	Positive or zero argument.	87.4	46	ROND
Logarithm, base e	LGEX1	Call LGEX1, DAC Arg 1	R=LGEX1(R)	$1/e >$ argument $\geq e$	931.2	60	LG2X1,MPY,ROND
*Logarithm, base e	LGEX2	Call LGEX2, DAC Arg 1	R=LGEX2(R)	$1/e >$ argument $\geq e$	111.4	59	LG2X2,ROND
Logarithm, base 2	LG2X1	Call LG2X1, DAC Arg 1	R=LG2X1(R)	$5 >$ argument ≥ 1	728.6	37	DIV,MPY
*Logarithm, base 2	LG2X2	Call LG2X2, DAC Arg 1	R=LG2X2(R)	$5 >$ argument ≥ 1	60.5	34	None
Multiply	MPY	Call MPY, DAC Arg 2.	R=R*R	None (overflow not possible.)	154.6	74	MPYS
Round up binary number	ROND	Call ROND, DAC Arg 1	R=ROND(R)	None	6.7	6	None
*Sine	SINX1	Call SINX1, DAC Arg 1	R=SINX1(R)	None	805.4	30	MPY
Sine	SINX2	Call SINX2, DAC Arg 1	R=SINX2(R)	None	47.0	25	None
Square root	SQRX1	Call SQRX1, DAC Arg 1	R=SQRX1(R)	None	675.8	61	MPY,DIV
*Square root	SQRX2	Call SQRX2, DAC Arg 1	R=SQRX2(R)	None	98.9	60	None
Floating-point:							
Absolute value	ABS	Call ABS, DAC Arg 1	R=ABS(R)	None	53.8	9	L$22,N$22
Add	A$22	Call A$22, DAC Arg 2	R=R+R	Overflow	238.1	184	ARG$,N$22,F$ER
Arctangent, principle value	ATAN	Call ATAN, DAC Arg 1	R=ATAN(R)	None	2332.8	228	ARG$,D$22,N$22,M$22,A$22,S$22
Arctangent, y/x	ATAN2	Call ATAN2, DAC Arg 2	R=ATAN2(R,R)	None	3647.0	228	ARG$,D$22,N$22,M$22,A$22,S$22
Convert (FORTRAN-generated) to double-precision.	DBLE	Call DBLE, DAC Arg 1	D=DBLE(R)	None	154.6	9	FAT,L22,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	293.8	8	L$22,C$21
Convert pair to complex	CMPLX	Call CMPLX, DAC Arg 2	C=CMPLX(R,R)	None	298.0	25	FAT,L22,H$22,L$55
Convert to complex format.	C$25	Call C$25, DAC Arg 1	C=C$25(R)	None	327.4	9	H$22,CMPLX
Convert to double-precision	C$26	Call C$26, DAC Arg 1	D=C$26(R)	None	10.6	8	None
Convert to integer	C$21	Call C$21, DAC Arg 1	I=C$21(R)	Integer > 15	252.5	24	N$22,A$22,F$ER
Divide	M$22	Call D$22, DAC Arg 2	R=R/R	Overflow, zero divisor.	1021.4	295	N$22,ARG$,RMPY,F$ER,DIV
Exponential, base e	EXP	Call EXP, DAC Arg 1	R=EXP(R)	Unnormalized argument, exponent overflow/underflow	4398.7	204	ARG$,N$22,M$22,S$22,A$22,D$22,F$ER
Hyperbolic tangent	TANH	Call TANH, DAC Arg 1	R=TANH(R)	None	5987.5	31	L$22,EXP,A$22,H$22,D$22
Load	L$22	Call L$22, DAC Arg 1	R=R	None	28.8	8	ARG$
Logarithm, base e	ALOG	Call ALOG, DAC Arg 1	R=ALOG(R)	Argument ≤ 0	3883.2	188	ARG$,C$12,A$22,MULT,M$22,S$22,F$ER
Logarithm, base 10	ALOG10	Call ALOG10, DAC Arg 1	R=ALOG10(R)	Argument ≤ 0	3888.0	188	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	1488.0	46	L$22,H$22,S$22,IFIX
Maximum value	MAX1	Call AMAX1, DAC Arg n	R=AMAX1(R1,R2,...Rn)	None	1201.9	46	L$22,H$22,S$22,IFIX
Minimum integer value	MINI	Call MINI, DAC Arg n	I=MINI(R1,R2,...Rn)	None	1494.7	47	L$22,H$22,S$22,IFIX
Minimum value	MINI	Call AMINI, DAC Arg n	R=AMINI(R1,R2,...Rn)	None	1208.6	47	L$22,H$22,S$22,IFIX

*Operates with multiply/divide option only.

Table 7-1. (Cont)
Mathematical Routines

Function	Routine	Calling Sequence	Mode	Errors	Timing (μsec)	Storage (Decimal)	Other Routines Used
Single-precision - (Cont)							
Floating-point - (Cont)							
Multiply	M$22	Call M$22, DAC Arg 2 . . .	R=R*R	Overflow	437.8	295	N$22,ARG$,RMPY,F$ER,DIV
Positive difference	DIM	Call DIM, DAC Arg 2 . . .	R=DIM(R,R) . . .	None	297.6	15	L$22,S$22
Raise to double-precision power.	E$26	Call E$26, DAC Arg 2 . . .	D=R***D	None	34263.4	17	FAT,C26,H$66,DLOG,M$66, DEXP
Raise to integer power	E$21	Call E$21, DAC Arg 2 . . .	R=R**I	None	R^2 499.2	47	ARG$,M$22,D$22
Raise to single-precision power.	E$22	Call E$22, DAC Arg 2 . . .	R=R**R	None	$R^4$522.6	29	ARG$,ALOG,M$22,EXP
Remainder	AMOD	Call AMOD, DAC Arg 2 . . .	R=AMOD(R,R) . .	None	2309.8	24	L$22,D$22,AINT,M$22,N$22, A$22
Sine, Cosine	SIN, COS .	Call SIN (or COS) DAC Arg 1.	R=SIN(R) or R=COS(R) .	None	4442.9	140	ARG$,N$22,M$22,S$22,A$22
Square root.	SQRT	Call SQRT, DAC Arg 1 . . .	R=SQRT(R). . .	Negative argument .	1545.6	71	ARG$,DIV$,D$22,A$22,F$ER
Store (hold).	H$22	Call H$22, DAC Arg 1 . . .	R=R	None.	33.6	13	ARG$
Subtract	A$22	Call S$22, DAC Arg 2 . . .	R=R-R	Overflow	241.9	184	ARG$,N$22,F$ER
Transfer sign of second argu- ment to first.	SIGN	Call SIGN, DAC Arg 2 . . .	R=SIGN(R,R) . .	None	71.0	20	L$22,N$22
Truncate fractional bits	AINT	Call AINT, DAC Arg 1 . . .	R=AINT(R) . . .	None	531.8	24	L$22,N$22,A$22,S$22
Twos complement	N$22	Call N$22, DAC Arg 1 . . .	R=N$22(R). . . .	None	8.6	10	None

*Operates with multiply/divide option only.

7-7

SECTION VIII
INPUT/OUTPUT LIBRARY

All routines in the DDP-516 Input/Output Library are described in this section. The purpose, storage requirements, and use (or calling sequence) is given for each routine. "Method", "Errors", and "Other Routines Used" are also included as applicable.

Standard I/O library mnemonics consist of four characters as follows.

a. The first character is generally either an I (for input) or an O (for output).

b. The second character (dollar sign) identifies the routine as a library routine.

c. The third character designates the device

    A ASR-33/ASR-35

    C Card reader or card punch

    P Paper tape reader or paper tape punch

    M Magnetic tape transport

d. The fourth character specifies the mode or function

    A ASCII

    B Binary

    E Eject (or end of file)

    H Heading

    I Initialize ·

    P Punch

    S End of message

    F File

The magnetic tape control and conversion mnemonics are an exception to the general rule for standard I/O library routines and are designated as follows.

CONTROL		CONVERSION	
C$MR	Rewind	C$6T08	IBM to ASCII
C$BR	Backspace record	C$8T06	ASCII to IBM ·
C$BF	Backspace file	M$UNIT	Physical to logical
C$FR	Forward record		
C$FF	Forward file		

Routines that are FORTRAN IV links are identified by F$ as the first two characters of the mnemonic.

FORTRAN IV INPUT/OUTPUT SUPERVISOR (F4-IOS)

Purpose

F4-IOS services all I/O requirements of the FORTRAN IV compiler. The compiler makes use of five entry points to F4-IOS as follows.

F4\$INT	Initialization
F4\$IN	Input
F4\$OUT	Binary output
F4\$SYM	Symbolic output
F4\$END	Compilation complete

This program will operate on a standard DDP-516 with a minimum of 8K storage.

Storage Requirements

F4-IOS requires $359_{10}(547_8)$ locations. Locations 100-113 are also initialized for use by the compiler.

Usage

Refer to the program listing, 3C Doc. No. 180016000 Rev. A, in the FORTRAN IV compiler manual for complete instructions on use.

INPUT/OUTPUT SUPERVISOR (IOS-16B)

Purpose

IOS-16B performs the necessary supervision and control of DAP-16 assembler input/output device requirements. IOS-16B consists of 15 entries that are used by the DAP assembler, as required. Each entry returns control to the assembler when its function has been accomplished.

Storage Requirements

IOS-16B requires $747_{10}(1353_8)$ locations.

Usage

Refer to program listing, 3C Doc. No. 180329000 Rev. A for complete details on operation and usage.

FORTRAN IV SCANNER AND CONVERSIONS ROUTINE (F$IO)

Purpose

F$IO accommodates the proper input/output device to accomplish the appropriate input/output conversion, as required, and to control the filling and emptying of the input/output buffer. This routine contains the argument transfer subroutine (F$AR), and the buffer closeout subroutine (F$CB).

Storage Requirements

F$IO requires $1255_{10}(2347_8)$ locations.

Usage

Refer to program listing, 3C Doc. No. 182618000 Rev. A, for complete instructions on use.

Other Routines Called

F$ER

ASR-33 TYPEWRITER INPUT DRIVER (F$R1)

Purpose

F$R1 provides linkage between the FORTRAN calling program and the I/O control subroutine (F$IO) and provides the driving logic needed to input on the typewriter keyboard by calling I$AA.

Storage Requirements

F$R1 requires $10_{10}(12_8)$ locations.

Calling Sequence

```
CALL    F$R1
DAC     N        (location of descriptor list given by FORTRAN format
                  statement)
```

Method

Refer to program listing, 3C Doc. No. 182610000 Rev. A, for details on the method used.

Other Routines Called

FIO, IAA

ASR-33 TYPEWRITER OUTPUT DRIVER (F$W1)

Purpose

F$W1 provides linkage between the FORTRAN calling program and the I/O control subroutine (F$IO). It also provides the driving logic needed to output on the ASR-33 typewriter by calling OAC, OAP, and O$AF.

Storage Requirements

F$W1 requires $61_{10}(75_8)$ locations.

Calling Sequence

```
CALL    F$W1
DAC     N        (location of format descriptor list given by
                  FORTRAN format statement)
```

Method

Refer to program listing, 3C Doc. No. 182611000, Rev. A, for details on method used.

Other Routines Called

FIO, OAC, OAP, OAF

PAPER TAPE READER INPUT DRIVER (F$R2)

Purpose

F$R2 provides linkage between the FORTRAN calling program and the I/O control subroutine (F$IO) and the driving logic needed to input from the paper tape reader by calling I$PA.

Storage Requirements

F$R2 requires $10_{10}(12_8)$ locations.

Calling Sequence

```
CALL    F$R2
DAC     N        (Location of the descriptor list given by FORTRAN format
                  statement, or 00000 if input is in binary format)

(Return)
```

Method

Refer to program listing, 3C Doc. No. 182612000 Rev. A, for details on method used.

Other Routines Called

F$IO and I$PA.

PAPER TAPE PUNCH OUTPUT DRIVER (F$W2)

Purpose

F$W2 provides linkage between the FORTRAN calling program and the I/O control subroutine (F$IO) and the driving logic needed to output on the paper tape punch by calling OPF, OPC, and O$PP.

Storage Requirements

This routine requires $46_{10}(56_8)$ locations.

Calling Sequence

```
CALL    F$W2
DAC     N       (Location of descriptor list given by FORTRAN format
                statement, or 00000 if output mode is binary)

(Return)
```

Method

Refer to program listing, 3C Doc. No. 182613000 Rev. A, for details on method used.

Other Routines Called

FIO, OPF, O$PC, and O$PP.

CARD READER INPUT DRIVER (F$R3)

Purpose

F$R3 provides linkage between the FORTRAN calling program and the I/O control subroutine (F$IO) and the driving logic needed to input from the card reader by calling I$CA or I$CB.

Storage Requirements

F$R3 requires $10_{10}(12_8)$ locations.

Calling Sequence

```
CALL    F$R3
DAC     N        (Location of descriptor list given by FORTRAN format
                 statement, or 00000 if input is in binary format)

(Return)
```

Method

Refer to the program listing, 3C Doc. No. 182614000 Rev. A, for details on method used.

Errors

Card reader error.

Other Routines Called

FIO, ICA

FORTRAN MAGNETIC TAPE INPUT DRIVER (F$R5-9)

Purpose

F$R5-9 controls reading of magnetic tape by connecting the FORTRAN calling program with the I/O control routine (F$IO) and the standard magnetic tape read subroutines.

Storage Requirements

F$R5-9 requires $81_{10}(121_8)$ locations.

Calling Sequence

```
CALL    F$Rx     (Where x=5, 6, 7, 8, or 9)
DAC     N        (Location of description list given by FORTRAN format
                 statement or 00000 if output mode is binary.)
```

Method

Refer to the program listing, 3C Doc. No. 180306000 Rev. A for details on method used.

FORTRAN MAGNETIC TAPE OUTPUT DRIVER (F$W5-9)

Purpose

F$W5-9 controls writing of magnetic tape by connecting the FORTRAN calling program with the I/O control routine (F$IO) and the standard magnetic tape write subroutines.

Storage Requirements

F\$W5-9 requires $58_{10}(72_8)$ locations.

Calling Sequence

CALL	F\$Wx	(Where x 5, 6, 7, 8, or 9)
DAC	N	(Location of descriptor list given by FORTRAN format statement, or 00000 if output mode is binary.)

Method

Refer to the program listing, 3C Doc. No. 180307000 Rev. A, for details on method used.

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

Purpose

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 will occupy bit positions 1-8 of the same word. The character originally occupying bit positions 7-12 will occupy bit positions 9-16.

Storage Requirements

C\$6T08 requires $63_{10}(77_8)$ locations

Calling Sequence

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

Method

Refer to the program listing, 3C Doc. No. 180091000, Rev C for details on the method used.

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

Purpose

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 will, after conversion, occupy bit positions 1-6 of the same word. The character originally occupying bit positions 9-16 will occupy bit positions 7-12. Bit positions 13-16 of each word will be set to zero.

Storage Requirements

C$8T06 requires $64_{10}(100_8)$ locations.

Calling Sequence

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

Method

Refer to the program listing, 3C Doc. No. 180082000 Rev B for details on method used.

VARIABLE INPUT DRIVER SELECTION (F$RN)

Purpose

F$RN controls the input devices for variable input device numbers. The value of the input device number is checked for correct limits and then used to determine the entry position of a jump table. The jump table transfers to the proper F$R - subroutine.

Storage Requirements

F$RN requires $31_{10}(37_8)$ locations.

Calling Sequence

```
LDA     D          (location of device number)
CALL    F$RN
DAC     N          (location of descriptor list given by FORTRAN statement,
                    or 00000 if format is binary)
(Return)
```

Errors

$d < 1$, or $d > 9$. Characters IO will be typed. Computer will stop. To continue, enter acceptable value in Register A and press START pushbutton.

Other Routines Called

F$R1 through 3, and F$R5 through 9.

VARIABLE OUTPUT DRIVER SELECTION (F$WN)

Purpose

F$WN controls the output drivers for variable output device numbers. The value of the output device number is checked for correct limits and then used to determine the entry position of a jump table. The jump table transfers to the proper F$W - subroutine.

Storage Requirements

F$WN requires $27_{10}(33_8)$ locations

Calling Sequence

```
LDA     D            (location of device number)
CALL    F$WN
DAC     N            (location of descriptor list given by FORTRAN format
                      statement, or 00000 if format is binary)
```

Errors

$d < 1$, or $d > 9$. Characters IO will be typed. Computer will stop. To continue, enter an acceptable value into Register A and depress the START pushbutton.

Other Routines Called

F$R1 through 9.

ASR-33 TAPE READER, ASCII (IAA, IAI)

Purpose

I$AA reads ASCII paper tape, using the ASR-33 paper tape reader. If I$AA is not initialized by I$AI, it will assume that the input buffer is 40 words long and that there are three tab settings corresponding to character positions 6, 12, and 30 (DAP 516 source format).

Storage Requirements

I$AA requires $127_{10}(177_8)$ locations.

```
                   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, 3C Doc. No. 189001000, Rev C, for details on the method used.

ASR-33 TAPE READER, BINARY (IAB, IABI)

Purpose

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

Storage Requirements

I$AB requires $121_{10}(171_8)$ locations.

Calling Sequence

```
Initialization  -  CALL    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 will halt. Check input tape for correct control characters between records. Press START pushbutton to take normal return.

Method

Refer to the program listing, 3C Doc. No. 189002000, Rev. B, for details on the method used.

ASR-33 TYPEWRITER CONTROL PACKAGE (OAP, OAC, O$AF)

Purpose

This routine performs one of three ASR-33 control functions depending on which entry is used. The entry mnemonics and corresponding control functions are as follows.

O$AP	Type a line
O$AC	Carriage return
O$AF	Advance to next line

Storage Requirements

This routine requires $54_{10}(66_8)$ locations.

Calling Sequence

(1)	CALL	O$AP	
	DAC	ARG1	(Where ARG1 is the location of the buffer to be typed out)
(2)	CALL	O$AC	
(3)	CALL	O$AF	

Method

Refer to the program listing, 3C Doc. No. 180255000, Rev. A for details on method used.

ASR-33 TYPEWRITER - LISTING AND HEADING ROUTINES (OLL, OHH)

Purpose

These routines type out listings on the ASR-33 typewriter. 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 Doc. No. 180774000 Rev B for a program listing.

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	(Head Heading Address)
	(Normal return)	

ASR-33 TAPE PUNCH, ASCII (OAA, OAI, OAS, OALDR)

Purpose

O$AA punches ASCII paper tape using the ASR-33 paper tape punch. This routine will assume, if not initialized by O$AI, that the data buffer is 40 words long and that there are three tab positions corresponding to character positions 6, 12, and 30 (DAP-516 source format). The O$AS entry is used to punch end of message, and the O$ALDR entry is used to punch 10 inches of blank tape.

Storage Requirements

These routines require 121_{10} (171_8) locations.

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 message -	CALL	O$AS
Leader -	CALL	O$ALDR

Method

Refer to program listing, 3C Doc No. 189003000, Rev C for details on method used.

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

Purpose

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

Storage Requirements

O$AB requires $87_{10}(127_8)$ locations.

Calling Sequence

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

Refer to program listing, 3C Doc. No. 189004000, Rev B for complete details on use.

PAPER TAPE READER, ASCII (IPA, IPI)

Purpose

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 will assume that the input buffer is 40 words long and that there are three tab settings corresponding to character positions 6, 12, and 30 (DAP-516 source format).

Storage Requirements

I$PA and I$PI require $117_{10}(165_8)$ locations.

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
                    DAC            (Data buffer address)
                    (End of message return)
                    (Normal return)
```

Refer to program listing, 3C Doc. No. 189006000, Rev D for complete information on use.

PAPER TAPE READER, BINARY (IPB, IPI)

Purpose

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

Storage Requirements

I$PB requires $109_{10}(155_8)$ locations.

Calling Sequence

```
Initialization  ·   CALL          I$PI
                    DAC            (Data)
                    (End of message return)
                    (Normal return)
Data read       –   CALL          I$PB
                    DAC            (Binary data address)
                    (End of message return)
                    (Normal return)
```

Errors

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

Method

Refer to the program listing, 3C Doc. No. 189007000, Rev B for details on the method used.

PAPER TAPE PUNCH CONTROL PACKAGE (OPP, OPC, O$PF)

Purpose

These routines perform one of three paper tape punch control functions depending on which entry is used. The entry mnemonics and corresponding control functions are as follows.

O$PP	Punch a line
O$PG	Punch carriage return
O$PF	Advance to next line

Storage Requirements

These routines require $51_{10}(63_8)$ locations.

Calling Sequence

CALL	O$PP	
DAC	ARG1	(Where ARG1 is the location of the buffer to be punched out)
CALL	O$PC	
CALL	O$PF	

Method

Refer to the program listing, 3C Doc. No. 180257000 Rev. A, for details on the method used.

PAPER TAPE PUNCH, ASCII (OPA, OPI, OPS, OPLDR)

Purpose

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 will assume that the data buffer is 40 words long and that there are three tab settings corresponding to character positions 6, 12, and 30 (DAP-516 source format).

Storage Requirements

These routines require $120_{10}(170_8)$ locations.

Calling Sequences

Data	- CALL	O$PA
	DAC	(Data buffer address)
	(Normal return)	

Initialization	- CALL	O$PI
	DEC	(Number of words in data register)
	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$PS

Leader - CALL O$PLDR

Refer to the program listing, 3C Doc. No. 189008000 Rev. C for complete details on use.

PAPER TAPE PUNCH, BINARY (OPB, OPS)

Purpose

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

Storage Requirements

O$PB requires $88_{10}(130_8)$ locations.

Calling Sequence

Data	- CALL	O$PB
	DAC	Data (Binary data address)
	(Normal return)	

End of message	- CALL	O$PS
	(Normal return)	

Refer to the program listing, 3C Doc. No. 189009000 Rev. B, for complete details on use.

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

Purpose

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 3C Doc. No. 181479000 for a program listing.

Storage Requirements

O$PL and O$PH require $141_{10}(215_8)$ locations.

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 (I$CA)

Purpose

I$CA reads ASCII (Hollerith) cards using the DDP-516 card reader. One card will be read on each I$CA entry. The data will be 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.

Storage Requirements

I$CA requires $75_{10}(113_8)$ locations.

Calling Sequence

	CALL	I$CA
	DAC	(Data buffer address)
	(End of file return)	
	(Normal return)	

Errors

Card reader empty, jammed, or in manual.

Method

Refer to the program listing, 3C Doc. No. 189011000 Rev. C, for details on method used.

CARD READER, BINARY (I$CB)

Purpose

I$CB reads column binary cards using the DDP-516 and reader.

Storage Requirements

This routine requires $42_{10}(52_8)$ locations.

Calling Sequence

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

Errors

Card reader empty, jammed, or in manual.

Method

Refer to the program listing, 3C Doc. No. 180609000 Rev. B, for details on method used.

MAGNETIC TAPE READ PACKAGE (IMA, IMB, I$MC)

Purpose

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

    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

Storage Requirements

These routines require $104_{10}(150_8)$ locations.

Calling Sequence

CALL	I$Mx	(where x is A, B, or C)
DAC	BUFA	(Buffer address)
DEC	WC	(Word count)
DEC	N	(Logical type unit)

(Record unreadable return)

(End of tape return)

(End of file return)

(Normal return)

Method

Refer to the program listing, 3C Doc. No. 182604000 Rev. B, for details on the method used.

Other Routines Called

M$UNIT

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

Purpose

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

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

Storage Requirements

This routine requires $67_{10}(103_8)$ locations.

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, 3C Doc. No. 182606000 Rev. B, for details on the method used.

Other Routines Called

M$UNIT

MAGNETIC TAPE WRITE PACKAGE (OMA, OMB, OMC, OME)

Purpose

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 as follows.

       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

Storage Requirements

These routines require $101_{10}(145_8)$ locations.

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)

)

(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

Method

Refer to the program listing, 3C Doc. No. 182605000 Rev. B, for details on the method used.

Other Routines Called

M$UNIT

MAGNETIC TAPE UNIT CONVERSION ROUTINE (M$UNIT)

Purpose

M$UNIT provides a physical tape number associated with a logical tape when called by the magnetic tape read, write and control routines.

Storage Requirements

This routine requires a variable number of locations depending upon the system configuration. (Usual size is 15_{10}.)

Calling Sequence

(Load logical tape number in Register A - bits 13-16.)

CALL   M$UNIT

(Physical tape number will be stored in Register A - bits 14-16.)

NOTE

The logical tape number must be between 1 and 8.
The physical tape number will be between 0 and 7.

Method

Refer to the program listing, 3C Doc. No. 180228000, Rev. B, for details on the method used.

FORTRAN MAGNETIC TAPE BACKSPACE DRIVER (F$F5-9)

Purpose

F$F5-9 controls backspacing of magnetic tape by connecting the FORTRAN calling program with the I/O control routine (F$IO) and the standard magnetic tape backspacing subroutines.

Storage Requirements

F$F5-9 requires $42_{10}(52_8)$ locations.

Calling Sequence

    CALL   F$Fx   (Where x= 5, 6, 7, 8, 9, or n.)

Method

Refer to the program listing, 3C Doc. No. 180310000 Rev. A, for details on method used.

FORTRAN MAGNETIC TAPE END-OF-FILE DRIVER (F$D5-9)

Purpose

F$D5-9 controls writing end-of-file marks on magnetic tape by connecting the FORTRAN calling program with the I/O control routine (F$IO) and the standard magnetic tape end-file subroutine.

Storage Requirements

F$D5-9 requires $27_{10}(33_8)$ locations.

Calling Sequence

    CALL   F$Dx   (Where x=5, 6, 7, 8, 9, or n.)

Method

Refer to the program listing, 3C Doc. No. 180308000 Rev. A, for details on method used.

FORTRAN MAGNETIC TAPE REWIND DRIVER (F$B5-9)

Purpose

F$B5-9 controls rewinding of magnetic tapes by connecting the FORTRAN calling program with the standard magnetic tape rewind subroutine.

Storage Requirements

F$B5-9 requires $27_{10}(33_8)$ locations.

Calling Sequence

      CALL   F$Bx   (Where x 5, 6, 7, 8, 9, or n.)

Method

Refer to the program listing, 3C Doc. No. 180309000 Rev. A, for details on method used.

SECTION IX
ERROR MESSAGES

LOADING MESSAGES

A message, which indicates the reason for halting, is always typed whenever loading halts. See Table 5-1 for an explanation of these messages.

DAP-16 ASSEMBLY PROGRAM

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. Examples of errors that are detected are shown in Table 9-1.

Errors in a field will generally result in that field being assembled as a 0. In the case of multiply defined symbols, the first symbol definition is used.

Table 9-1.
Error Messages Generated by the DAP-16 Assembly Program

Error Message	Condition
A	Address field missing where normally required, or error in address format
C	Erroneous conversion of a constant or a variable field in improper format
F	Major formatting error
L	Location symbol missing where required, or error in location symbol
M	Multiply defined symbol
N	Missing name (main program or subroutine)
O	Operation code missing or in error
R	Relocation assignment error
S	Address of variable field expression not in sector being processed or sector zero (applicable only in load mode)
T	Improper use of or error in index field
U	Undefined symbol
V	Unclassified error in variable field of multiple field pseudo-operator (i.e., DEC, OCT, etc.)
X	Symbol table or literal table exceeded

FORTRAN COMPILER PROGRAM

Error Message

Any time the compiler detects an error in the format of a FORTRAN statement, a two line error message is typed or printed in the listing. If the error is of a type that is recognized as soon as it is encountered, the first line is a duplicate of the line in which the error occurred. If the error cannot be recognized until later in the program, the line at which the error is recognized contains a left pointing arrow (←) in column 6. In either case, the second extra line consists of a row of asterisks broken by the word ERROR in the left-hand margin and an error diognostic in approximately the same horizontal position as the error.

The diagnostic messages generated by the FORTRAN IV compiler program to indicate coding errors are listed in Table 9-2. Some errors are recoverable and their presence will not prevent execution of the program. In most cases, however, a program containing such errors will generate incorrect or ambiguous results. It is recommended that all coding errors be corrected before a program is run.

Table 9-2.
Error Messages Generated by the FORTRAN Compiler Program

Error Message	Condition
AE	Arithmetic statement function has over 10 arguments
AG	Subroutine or array name not in an argument
AR	Item not an array name
BD	Code generated within a block data subprogram
BL	Block data not first statement
CE	Constant's exponent exceeds 8 bits (over 255)
CG	Compiler or computer error caused a jump to 00000
CH	Improper terminating character (punctuation)
CM	Comma outside parentheses, not in a DO statement
CN	Improper constant (data initialization)
CR	Illegal common reference
DA	Illegal use of a dummy argument
DD	Dummy item appears in an equivalence or data list
DM	Data and data name mode do not agree
DT	Improper DO termination
EC	Equivalence group not followed by comma or CR (carriage return)
EQ	Expression to left of equals, or multiple equals
EX	Specification statement appears after cleanup
FA	Function has no arguments
FD	Function name not defined by an arithmetic statement
FR	Format statement error
FS	Function/subroutine not the first statement
HF	Hollerith character count equals zero
HS	Hollerith data string extends past end of statement
IC	Impossible common equivalencing
ID	Unrecognizable statement
IE	Impossible equivalence grouping
IF	Illegal IF statement type
IN	Integer required at this position
IT	Item not an integer
MM	Mode mixing error
MO	Data pool overflow
MS	Multiply defined statement number
NC	Constant must be present
ND	Wrong number of dimensions
NF	No reference to format statement
NR	Item not a relative variable
NS	Subprogram name not allowed
NT	Logical NOT, not an unary operator

Error Message	Condition
NU	Name already being used
NZ	Non-zero string test failed
OP	More than one operator in a row
PA	Operation must be within parentheses
PH	No path leading to this statement
PR	Parentheses missing in a DO statement
PW	*Preceded by operator other than another*
RL	More than 1 relational operator in a relational example
RN	Reference to a specification statement's number
RT	Return not allowed in main program
SC	Statement number on a continuation card
SP	Statement name misspelled
ST	Illegal statement number format
SU	Subscript incrementer not a constant
TF	"Type" not followed by "Function" or list
TO	Assign statement has word TO missing
UO	Multiple + or - signs, not as unary operators
US	Undefined statement number
VD	Symbolic subscript not dummy in dummy array or symbolic subscript appears in a non-dummy array
VN	Variable name required at this position

LIBRARY SUBROUTINES

The subroutines in the standard FORTRAN IV library are designed for use with a machine configuration without the high-speed multiply and divide hardware option. Subroutine arguments are one of four types: integer, real, double, or complex.

If an error is detected by a subroutine, the object error diagnostic subroutine (F$ER) will be called. Normally, a two-character mnemonic will be typed on the ASR-33/35 and the computer will halt. If SENSE SWITCH 3 is on, F$ER will return control to the calling subroutine which will exit, with meaningless results, in most cases. Table 9-3 lists those subroutines that call on F$ER and their associated error message.

Table 9-3.
Error Messages Generated by the Library Subroutines

Error Message	Condition	Generating Subroutine
DA	Arithmetic Overflow	A$66/S$66 (Double Add/Sub)
DL	Negative or Zero Argument	DLOG/DLOG10 (Double Logarithm)
DM	Arithmetic Overflow or Zero Divisor	M$66/D$66 (Double Mult/Div)
EQ	Arithmetic Overflow	A$81 (Add Integer to Double Exponent)
EX	Exponent Overflow	EXP (Real Exponential)
FE	Format Error	F$IO (Format Scanner and Conversions)
II	Result Greater (2**15) -1	E$11 (Integer Raised to Integer)
IN	Input Error	F$IO (Format Scanner and Conversions)
RI	Exponent Greater than 15	C$21 (Convert Real to Integer)
SA	Arithmetic Overflow	A$22/S$22 (Real Add/Sub)
SD	Arithmetic Overflow or Zero Divisor	D$22 (Real Div)
SM	Arithmetic Overflow	M$22 (Real Multiply)
SQ	Negative Argument	SQRT (Real Square Root)

SECTION X
PAPER TAPE FORMATS

This section contains descriptions of the paper tape formats that are used as a principle input/output medium for the DDP-516 computer. Data is recorded on paper tape by groups of holes arranged in a definite format along the length of the tape. Paper tape is a continuous recording medium, as opposed to cards which are fixed in length, and the length of data records is limited only by the input/output requirements of the system. A vertical column of holes extending across the tape is referred to as a frame. A horizontal row of holes extending the length of the tape is referred to as a channel. For paper tapes punched and read by the DDP-516 system, there are eight channel-hole positions per frame, and one small sprocket hole. (See Figure 10-1.)
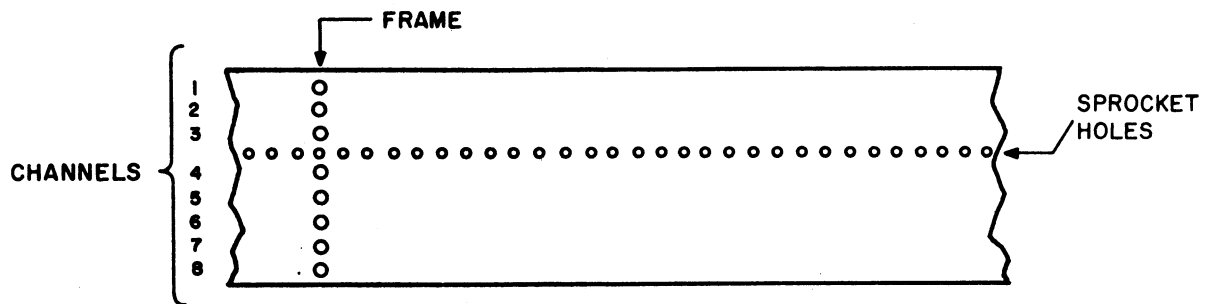

Figure 10-1. General Paper Tape Format

The format descriptions given in this section apply to tapes punched by the high-speed paper tape punch, as well as those punched by the ASR-33/35 paper tape punch. Paper tape formats used with DDP-516 systems fall into two main categories: an ASCII format (for punching source code) and a 4/6/6 format (for punching object code).

ASCII Format

ASCII format is an octal code that uses eight channels to define one character per frame. Each frame is read from channel 8 to channel 1 in bit groups of 2/3/3 as illustrated in Figure 10-2. Two ASR-33/35 typewriter control codes, '212 for line feed and '215 for carriage return, are represented in Figure 10-2 to illustrate use of ASCII format.
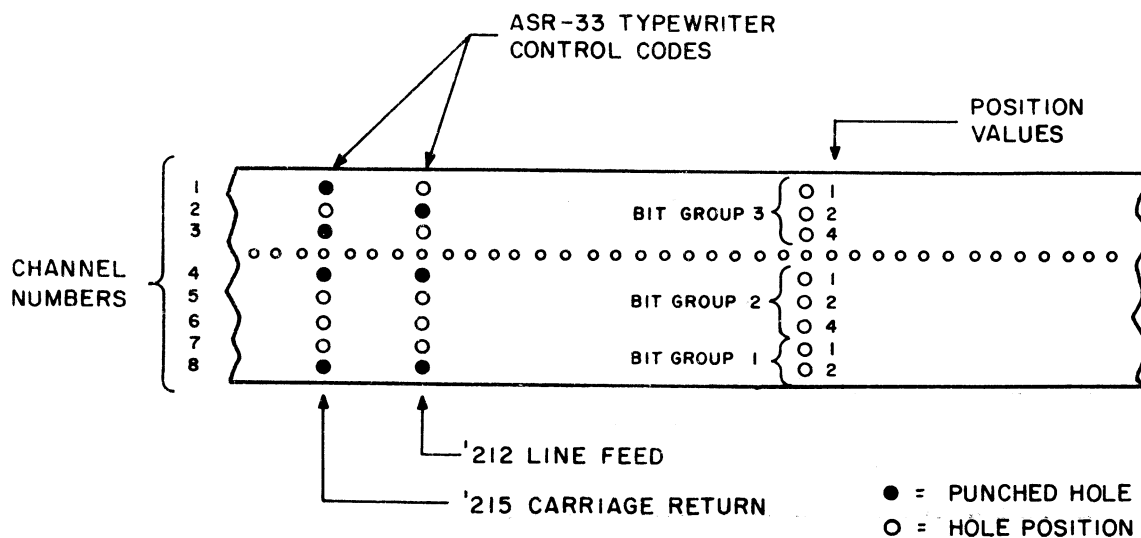
Figure 10-2. ASCII Format

Source Tape Preparation

A DAP-16 or FORTRAN source-program data line for the ASR-33 is recorded on paper tape in ASCII format as follows.

..LINE FEED ... TEXT ... X-OFF ... CARRIAGE RETURN...

The text string between the LINE FEED at the beginning of the line and the X-OFF at the end of the line is read into the input buffer. The LINE FEED, X-OFF, and CARRIAGE RETURN are control characters and are not input as part of the text string. The X-OFF at the end of the line (preceding the CARRIAGE RETURN) is necessary to be compatible with the ASR-33/35 input routines. (The ASR-35 requires a RUBOUT following the CARRIAGE RETURN.) If the tapes to be read by the paper tape read subroutine are never to be read by the ASR-33/35 tape read routine, the X-OFF (and RUBOUT) may be omitted.

When preparing a source tape using the ASR-33, depress the LINE FEED key, type the desired ASCII record (maximum of 72 characters), and then depress the X-OFF and CARRIAGE RETURN keys. Repeat this process for each record. If a character is punched erroneously, depress the back space and RUBOUT keys and proceed with the rest of the line. If a line is punched erroneously, depress the Left Arrow, X-OFF, and CARRIAGE RETURN keys. Source tapes punched using the ASR-35 are prepared in a similar manner except that the maximum number of characters per record is 75, and the RUBOUT key must be punched after the RETURN key.

Tabs may be used to compress the data, much as tabs are used on a typewriter. The backslash character (\) '334 is used as a tab code. The backslash is punched on the ASR-33/35 as an upper case L (FORM). Tabs may be used whenever a string of spaces precedes a TAB STOP. The tab is punched in place of the spaces. Another way to describe the backslash is that it is used as a field delimiter. For example, the backslash is ordinarily used when the location, operation, or variable field is not present.

The ASCII paper tape read subroutine (I$AA) will assume, if not initialized, that there are three tab positions corresponding to character positions 6, 12, and 30 (DAP-16 source code format).

The END OF MESSAGE (EOM) record has the following format.

> ... X-OFF EOM (EOM is '203 and is punched by using
>   the CONTROL KEY with the letter C).

The paper tape read subroutine will read one line of data per entry. If an end of message code ('203) is encountered at any point in the line, the end of message return will be taken, otherwise, the normal return is taken when a carriage return is encountered. The end of message code is usually used as a unit record following the last data line to be read.

If the data line exceeds the length of the data buffer, those characters in excess will be ignored. No error indication will be given. Similarly, if a tab is encountered after the last tab-stop has been passed, the tab will be treated as a space. No error indication will be given. The data line may be shorter than the length of the data buffer in which case, the buffer is filled out with spaces.

Figure 10-3 shows a portion of an actual assembly listing (DAP-Test Program) along with one line of corresponding source code punched on paper tape.

4/6/6 Format

The 4/6/6 format, in which object programs are punched is a binary code that uses three frames to define one 16-bit word. As illustrated in Figure 10-4, bits 1-4 of a 16-bit word are encoded to make up the first frame, 5-10 the second frame, and 11-16 the third frame. Control codes that are punched in ASCII format precede and follow each block of 16-bit words encoded in 4/6/6 format and use only one frame each. Each data block begins with an SOM ('201) and ends with a DC4 (or X-OFF '223).

When a frame is punched, channels 6 and 7 are ordinarily left blank. This is done so that a frame will not print when read by the ASR. However, certain six-bit patterns correspond to control functions, which if executed by the ASR, would interfere with further reading of this tape. Therefore, these (eight) six-bit groups are translated according to Table 10-1 before being punched. A tape punched using this translation will not type anything when read in by the ASR. Hence, the 4/6/6 format is sometimes called the "Invisible Format."
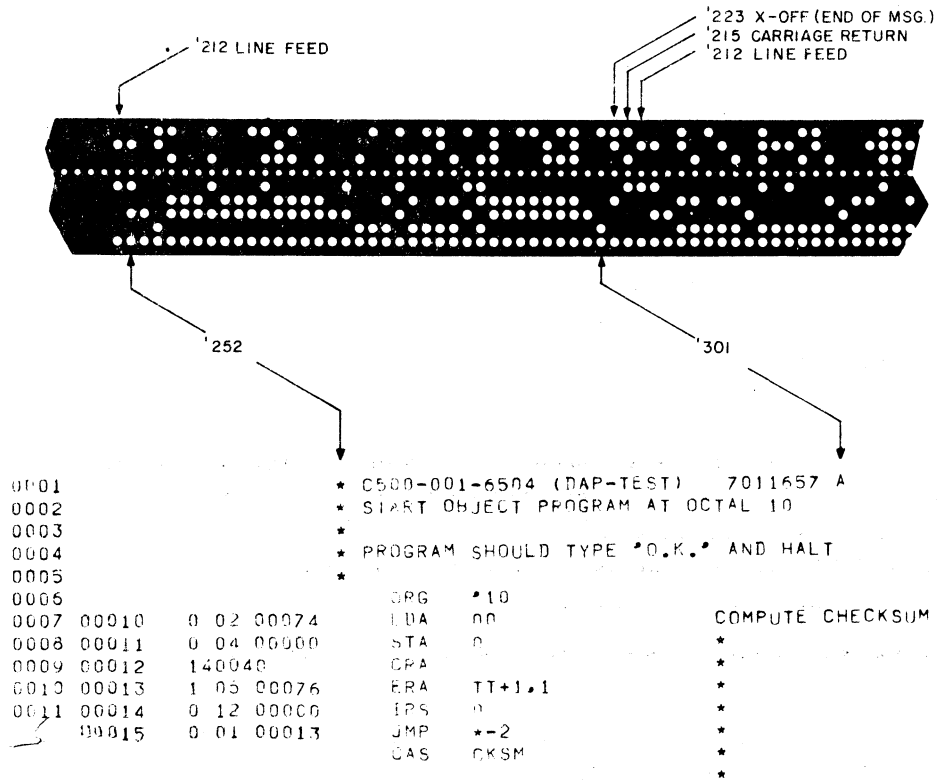
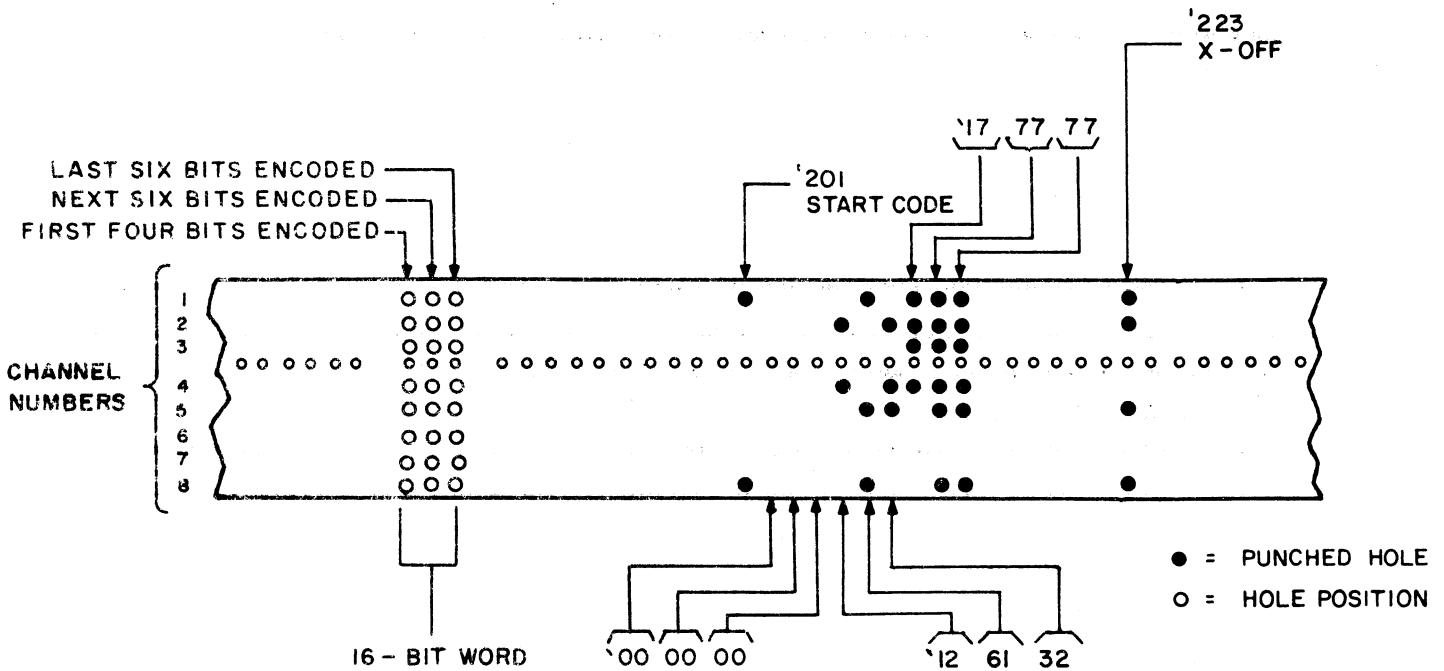Figure 10-3. DAP-16 Source Code Punched in ASCII

Figure 10-4. 4/6/6 Format

Table 10-1.
4/6/6 Translations

Intended 4 or 6 Bits	Is Converted to Frame	Possible Ambiguity with ASR-33/35 Control
05 or 45	174 or 374, respectively	WRU
12 or 52	175 or 375	LF
21 or 62	176 or 376	X-ON
23 or 63	177 or 377	X-OFF

In Figure 10-5, the instructions at address 00024 (data word 13 on the tape) when encoded without translation would result in an '45 being read by the ASR-33/35. The '45 is the WRU control code and would trigger the answer-back drum. This ambiguity is resolved however by the translation. (Note in Figure 10-5 that channels 6 and 7 are punched in the translation.)
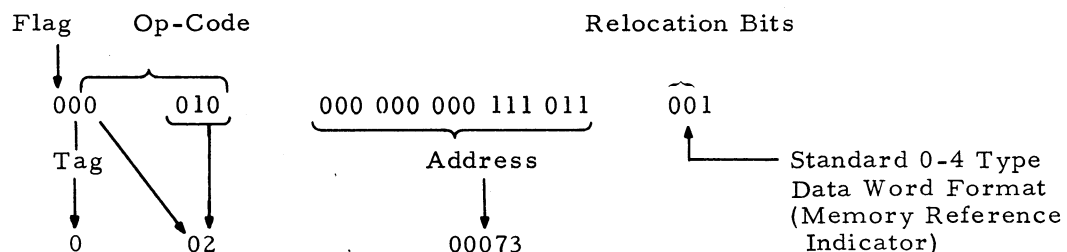
Object tapes that are produced by both the DAP assembly and the FORTRAN compiling process are punched in the binary 4/6/6 format. Within each block of object code, there is a variable-length sequence of data words. There are a number of different types of block, and they are defined in Section IV of the DAP-16 Manual, 3C Doc. No. 130071629. Object tapes generated in the 4/6/6 format are accepted by and compatible with both the DDP-516 standard and expanded loaders.

Figure 10-5 shows a portion of an actual assembly listing (DAP-TEST Program) along with corresponding object code punched in paper tape. As explained in the DAP Manual, the first 4/6/6 group after the start code defines the block type. This particular block type is 0-4 (000400), or a data block. The next 4/6/6 group specifies the number of data words in the data block. In this case there are 72_8 words in the data block. The words are interpreted as per the format for this block type given in the DAP Manual. In block type 0-4 the remaining bits are grouped into groups of 24 bits each. These groups are formed by using 16 bits of one 4/6/6 group and the first 8 bits of a second 4/6/6 group. The remaining 8 bits of the second 4/6/6 group and 16 bits of a third 4/6/6 group are used to form the second 24-bit group.

In the example given in Figure 10-5, the first 24-bit group is

$$000010000000000111011001$$

The DAP-16 Manual (3C Doc. No. 130071629) further explains the meaning of each 24-bit group, based on the last three bits. This one converts to the LDA instruction at address 00010 as follows.

Flag   Op-Code   Relocation Bits

000   010   000 000 000 111 011   001

Tag   Address   Standard 0-4 Type
                                          Data Word Format
0   02   00073   (Memory Reference
                                          Indicator)

DATA WORDS

1 & 2 | 3 & 4 | 5 & 6 | 7 & 8 | 9 & 10 | 11 & 12 | 13 & 14

'201 START CODE
BLOCK TYPE
NO. OF WORDS
FIRST LOCATION

FIRST DATA WORD

/000/010/000/000/000/111/011/001

FT OP-CODE   ADDRESS

0   02   00073

SECOND DATA WORD

000/100/000/000/000/000/000/001

FT OP-CODE   ADDRESS

0   04   000000

```
0001                          *  C500-001-6504 (DAP-TEST)    7011657 A
0002                          *  START OBJECT PROGRAM AT OCTAL 10
0003                          *
0004                          *  PROGRAM SHOULD TYPE 'O.K.' AND HALT
0005                          *
0006                          *
0007                             ORG    '10
0008  00010   0 02 00073        LDA    00               COMPUTE CHECKSUM
0009  00011   0 04 00000        STA    0                *
0010  00012   140040            CRA                     *
0011  00013   1 05 00075        ERA    TT+1,1           *
0012  00014   0 12 00000        IRS    0                *
0013  00015   0 01 00013        JMP    *-2              *
0014  00016   0 11 00076        CAS    CKSM             *
0015  00017   000000            HLT                     *
0016  00020   0 01 00022        JMP    *+2              *
0017  00021   000000            HLT                     WRONG SUM
0018  00022   0 02 00041 T      LDA    *-3              RIGHT SUM
0019  00023   0 04 00000        STA    0                TYPE 'O.K.'
0020  00024   1 02 00045 TTTT   LDA    MSG+3,1          *
0021  00025   34 0104           SKS    '104             *
              31 00025                 *-1              *
```
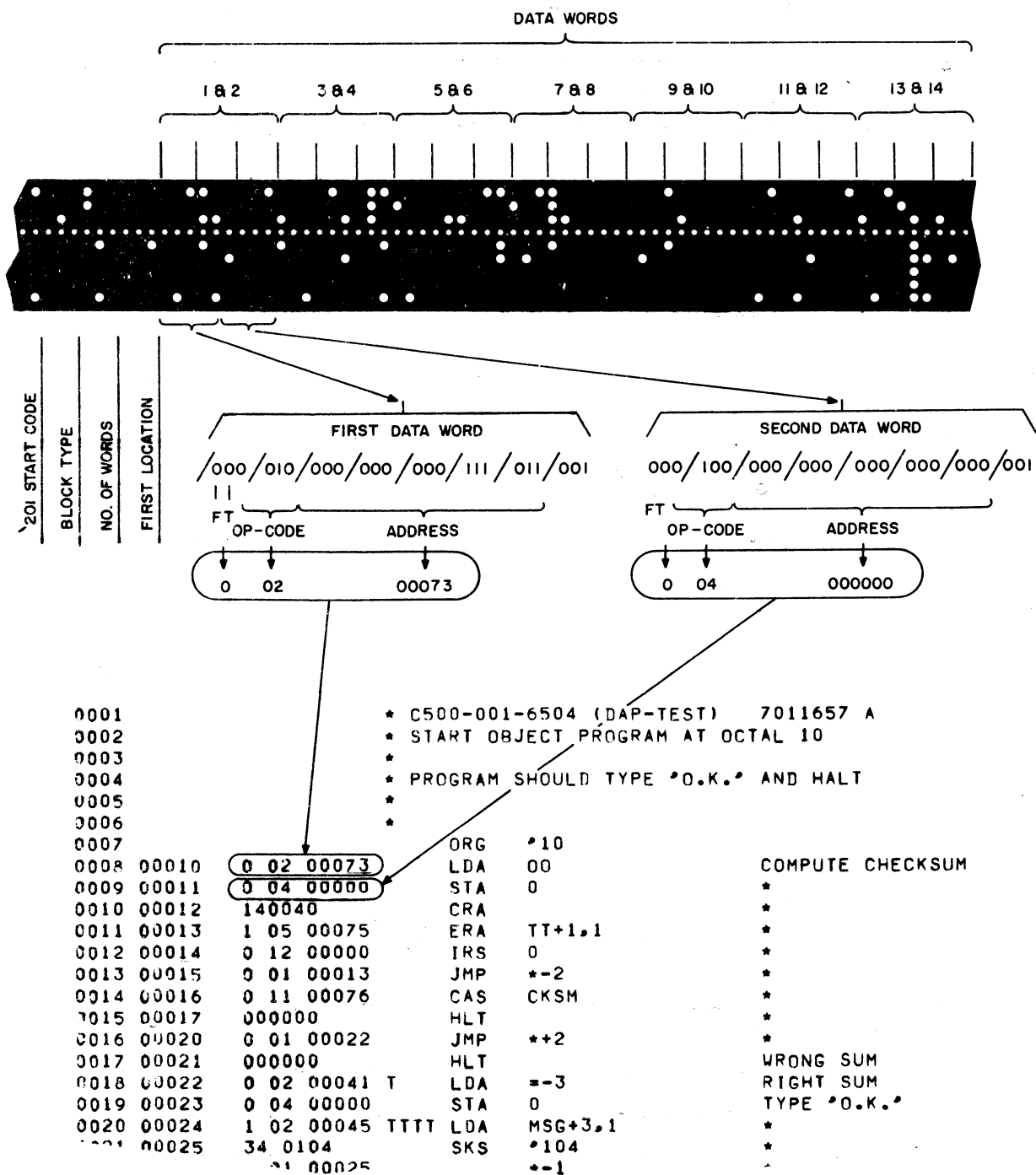
Figure 10-5.   DAP-16 Object Code Punched in 4/6/6 (Invisible) Format