PDP-9

UTILITY PROGRAMS

ADVANCED SOFTWARE SYSTEM

Programmer's Reference Manual

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

CONTENTS

To The Reader

Notification of changes and revisions to this document, to the software described, and of new software products available from the DEC Program Library, is currently published in DECUSCOPE, the magazine of the Digital Equipment Computer User's Society (DECUS). This information appears in a section of DECUSCOPE called "DEC Library News."

Revised software products and documents are shipped only after the Program Library receives a specific request from a user.

DECUSCOPE is distributed periodically to both DECUS members and to non-members who request it. If you are not now receiving this information, you are urged to return the request form below so that your name will be placed on the mailing list.

To: DECUS Office,
      Digital Equipment Corporation,
      Maynard, Mass. 01754

□   Please send DECUS installation membership information.

□   Please send DECUS individual membership information.

□   Please add my name to the DECUSCOPE non-member mailing list.

Name _____

Company _____

Address _____

_____

(Zip Code)

DDT-9

CONTENTS

APPENDIXES

SECTION 1
INTRODUCTION

1.1   GENERAL INFORMATION

DDT-9 (the Dynamic Debugging Technique program for the PDP-9) provides convenient on-line debugging assistance for MACRO-9 and FORTRAN programmers. By typing simple commands on the Teletype keyboard, programmers may make corrections and additions in symbolic code (or octal), suspend execution of the program at any predetermined point during the debugging run, and examine the status of any memory word in the program. The user's program is started and stopped by commands to DDT-9. Under normal conditions, the user is always able to stop a "runaway" program.

DDT-9 operates as part of the PDP-9 Advanced Software System. It is loaded into memory (the top 1600_{10} positions) along with the Linking Loader which, upon command, loads the user's program (including the symbol table and any sub-programs) and the needed I/O handlers, FORTRAN Object Time System routines and library subroutines. DDT-9 disables the automatic priority interrupt immediately upon receiving control from the Monitor.

All user communication with DDT-9 is via the Teletype, which may be any model included in standard PDP-9 configurations.

DDT-9 interprets all numeric input, and outputs all numeric data in octal radix. The digits 8 and 9 are treated as alphabetic characters.

1.2   TERMINOLOGY USED IN THIS MANUAL

)   A non-printing character used for text representation of the carriage return key.

    ↓   A non-printing character used for text representation of the line feed key.

    ↑   A text representation of the control key, always used in conjunction with another key. It is also the printing character, up arrow.

    ↑T   The non-printing character obtained by holding the control key while striking the T.

           The term C (R) represents the content of storage word R.

           In examples, underlining designates information typed by DDT-9.

    +   Elements are to be added.

    −   Elements are to be subtracted.

    ⊔   (Space) field delimiter, as between operation code and address.

A Transfer Vector is a word which contains the 15-bit address of another word. Bits 0-2 are meaningless and may be used for codes. In the PDP-9, transfer vectors are used in indirect addressing, by the Linking Loader for subroutine calls, and are required in addressing to another memory bank.

SECTION 2
DEBUGGING WITH DDT

2.1   LOADING THE PROGRAM

In an I/O Monitor (paper tape) environment, the Linking Loader forms an integral part of the DDT tape.

In the Keyboard Monitor, the teletype command DDT (DDTNS) calls the Linking Loader as well as DDT. (DDTNS is used to prevent loading of the user symbol table to save memory.)

The first response from the teletype, in either system, will be:

LOADER

>

The user program is then loaded in the usual manner. (See section 2.1 and 2.2 of the Linking Loader manual.) When loading is complete, DDT takes control and types:

DDT

>

to indicate its readiness to accept DDT commands.

With the Keyboard Monitor, DAT slots -4 (user program) and -5 (user external library, if any) must be assigned to appropriate devices for proper loading. DAT slot -6 (patch file output) and -10 (patch file input) must be assigned to the paper tape punch and paper tape reader, respectively, if the patch file capability is to be used. Otherwise DAT slots -6 and -10 should be assigned to NONE.

2.2   USING THE BREAKPOINTS

A breakpoint provides a convenient means of interrupting a user program at any predetermined step, allowing examination of the program status at that point. DDT-9 inserts a breakpoint (upon request) by replacing the indicated instruction with a jump to DDT-9. When the program reaches that point, control shifts to DDT-9, which types the number of breakpoint, the address of the breakpoint, the contents of the AC, the status of the Link, and the go-ahead signal (>). The user may then perform any of the debugging operations explained in this manual.

DDT-9 allows the use of four breakpoints to facilitate debugging when there is uncertainty as to which path the program will follow.

The user may place a breakpoint at any point in his program, considering only the following limitations:

a.   Instructions which are program modified

b.   Instructions which are used as literals

c.   XCT instructions pointing to memory reference instructions.

Breakpoints may be placed on skip, jump, and JMS instructions. Breakpoints may also be placed on CAL instructions, but since CAL instructions may contain arguments required by the called subroutine, as well as having a variable number of subsequent arguments, DDT-9 is unable to simulate the CAL (as it is able to simulate a JMS). Therefore, a breakpoint which has been placed on a CAL is removed by DDT-9 before continuing (exclamation point command). However, DDT-9 retains the request for a breakpoint at that location, and restores it if another breakpoint is entered and exited. If the user wants to place a breakpoint at a CAL, and restore it after each stop, he could place a second break-point at the return from the CAL, as shown in this example.

```
        LOC     CAL 3          .        (Breakpoint 1)
                12
                LAC BUFF                (Breakpoint 2)
```

Upon leaving the second breakpoint at LOC + 2, the breakpoint on the CAL instruction is restored.

Operation of breakpoints requires one auto-index register; DDT-9 initially assumes register 17. The user may specify any other auto-index register by modifying DDT-9's special register, AX$, as follows:

```
        AX$/    000017    10 )        (Modification procedure is explained later in
                                       this manual.)
```

The commands controlling breakpoints are as follows:

k n"	Causes a breakpoint to be inserted at location k. The number n (1-4) is assigned to that breakpoint.
n"	Causes the breakpoint assigned the number n to be removed.
"	Causes all existing breakpoints to be removed.

The insertion of a breakpoint takes place when control returns to the user program. The breakpoint occurs before execution of the instruction at the breakpoint address.

Examples:

LOC + 1 ⌴ 1"	Inserts a breakpoint at LOC+1
TAG ⌴ 2"	Inserts a breakpoint at TAG
1"	Removes breakpoint number 1

A breakpoint number may be reassigned without first removing the previous assignment.

To restart from a breakpoint, the user simply types an exclamation point (!). DDT-9 restores the AC and Link and returns control to the user's program, starting with the instruction at the breakpoint address. An octal number typed before the exclamation point will cause DDT-9 to bypass that breakpoint n times. This ability is convenient when a breakpoint has been inserted in a program loop, and the user does not wish to stop every time through the loop.

If the user's program does not reach the breakpoint, the operator may stop the action and return control to DDT-9 by typing control T (hold the CONTROL key down while striking the T). DDT-9 will type the go-ahead (>). The program interrupt control must be on to perform this operation.

2.3 EXAMINATION AND MODIFICATION

DDT-9 provides several variations of the procedure for examining and modifying the contents of any storage word. They are:

k/   The slash, typed after an address (k) causes the addressed storage word to be opened and its contents displayed on the teleprinter. For example,

   LOC/ TAD COUNT

where the instruction TAD COUNT is contained at the location labeled LOC. The storage word is now opened and may be modified by typing the desired content and issuing one of the commands described below.

⤶   The carriage return closes the storage word and resets DDT-9, enabling it to accept other commands. Any change which has been entered is incorporated, as shown below:

   LOC/   TAD COUNT   ⤶
   TAG/   JMP LOC   JMP   LOC+1 ⤶

↓   The line feed closes the storage word, then opens the next sequential storage word:

   LOC/   TAD COUNT   ↓
   LOC+1/   CMA

↑   The up arrow closes the storage word, then opens the preceding storage word.

   LOC/   TAD COUNT   ↑
   LOC −1/ LAC A

↑Z   Control Z allows the user to examine (and modify) a single storage word, out of sequence, and then return to the original sequence. This command closes the storage word, then opens the referenced storage word. A line feed will then open the next storage word in the original sequence, as shown:

   LABEL/   JMP LOC   ↑Z
   LOC/   TAD COUNT   TAD CNTR ↓
   LABEL+1/ LAC HOLD

↑A   Control A allows the user to examine a new sequence of storage words. This command closes the storage word, then opens the referenced storage word, establishing a new sequence. A line feed will then open the second storage word in the new sequence.

          LABEL/   JMP LOC   ↑A

          LOC/   TAD COUNT   TAD CNTR   ↓

          LOC+1/   CMA

↑X   Control X is used, in multi-memory bank systems in conjunction with transfer vectors, to examine a new sequence of storage words. This command operates with a 15-bit address taken directly from the currently open word. (In contrast, the ↑Z and ↑A operations take 13 bits from the currently open word and the two memory bank bits from the address of the open storage word.)

          TAG/   36307   X

          36307/   000000

2.4   TYPE-OUT MODES

DDT-9 allows the user to choose from several modes of representing the requested information. These modes, and their commands, are as follows:

NUM$ In this mode, DDT-9 types memory word contents as 6-digit octal numbers, including any leading zeroes.

TV$ In this mode, DDT-9 interprets words as transfer vectors. Bits 0-2 are ignored, and bits 3-17 are interpreted according to the address modes as described below.

SYM$ In this mode, which is assumed initially, DDT-9 interprets words as symbolic instructions. Bits 0-3 are first examined to determine the instruction class. If bit 4 (indirect addressing bit) of a memory reference instruction is set, an asterisk (*) is typed after the mnemonic op code. The address portion is handled according to the address mode as described below. Operate instructions are further examined for specific mnemonic codes. (See appendix 2 for recognized codes.) Operate instructions not found in DDT-9's table are typed out as NOP+XXXX. Subroutine calls, extended arithmetic element, and input/output instructions are interpreted as CAL+XXXX, EAE+XXXX, and IOT+XXXX, respectively.

:   The colon, typed after a word has been displayed in either numeric (NUM$) or symbolic (SYM$) mode, causes DDT-9 to retype the word in the alternate mode.

        LOC/   TAD LABEL   :   340126

   or   LOC/   340126   :   TAD LABEL

=   The equal sign, typed after a word has been displayed in either numeric or symbolic mode, causes DDT-9 to retype the word as a transfer vector.

        LOC/   CAL+126   =   LABEL

2.4.1   Address Modes

The following commands set the address mode, which affects the handling of transfer vectors, address portions of memory reference instructions, and display addresses.

REL$ In this mode, which is assumed initially, DDT-9 types addresses which are relative to user defined symbols.

        LOC/   TAD LABEL-3

If there is no symbolic label within $\pm77_8$ positions, the address is typed as relocatable (see next paragraph below). Symbols defined in direct assignments are not recognized by DDT-9.

RLC$ In this mode, DDT-9 types addresses in relocatable form, as shown on the assembly listing. For example,

        LOC/   TAD 147

ABS$ In this mode, DDT-9 types addresses in absolute form:

        LOC/   TAD 13147

The difference between the results of RLC$ and ABS$ modes is the relocation factor (in this case, 13000). The relocation factor is found in the memory map output by the Loader.

The user may type modification input in whatever representation he finds most convenient. There are, however, two points to keep in mind.

If a memory reference mnemonic is entered with a numeric address, DDT-9 assumes that address to be relocatable unless the address output mode has been set to ABS$. For example,

        LOC/   TAD COUNT   TAD 147 ↓

(DDT-9 adds the relocation factor before storing the information).

A requested address, typed numerically, is always considered absolute.

        41/   Opens word 41 of the machine.

        13000+41/ or 13041/   Opens word 41 of the program, where 13000 is the relocation factor.

2.5 STARTING AND RESTARTING

DDT-9 receives control, initially, from the Monitor and normally regains control from the user's program by means of a breakpoint, as described above. A control T may be typed at any time (if the program interrupt control is enabled) to restore control to DDT-9.

The following commands shift control from DDT-9 to the user's program:

' The apostrophe, typed alone, starts the user's program at its normal starting address. (That address given in the source .END statement, or the first physical location of the first program loaded.

k' The user may start his program at any other point by simply typing that address ahead of the apostrophe.

! The exclamation point restarts the user's program after a breakpoint. The AC and the Link are restored before continuing.

n! An octal number (n) entered before the exclamation point causes DDT-9 to bypass that breakpoint n times before stopping again. This ability is useful when a break-point has been placed in a program loop.

2.6 SEARCHING OPERATIONS

DDT-9 has a powerful searching operation with which every word in a user's program having particular characteristics can be found with ease. Two special locations, LO$ and HI$ (further ex-plained in the next section), control the limits of the search, and a mask (MSK$) allows the search to be based on all or any portion of the word. The mask is initially set at 777777, for a full word search; and the limits are initially set to encompass the entire user's program, including all subprograms and library routines.

There are three types of searches as follows:

k ⌣ EQ$ Starts a search for all words, within the set limits, whose contents, after mask-ing by C(MSK$), are equal to the expression k.

k ⌣ UN$ Starts a search for all words, within the set limits, whose contents, after mask-ing by C(MSK$), are not equal to the expression k.

k ⌣ ADR$ Starts a search for all memory reference instructions, within the set limits, with effective addresses which, after masking by C(MSK$), are equal to the address k. Indirect addressing is followed one step.

Examples:

> LOC+1 ⌐ ADR$ might produce

TAG/	LAC LOC+1
POS/	XOR LOC+1
LABEL/	DAC* POINT

If the > is typed with no other output, the search routine has found no qualifying words.

2.7 SPECIAL LOCATIONS USED BY DDT-9

The following special locations contain information of use to the user, and which he may wish to change.

AC$	Holds C(AC) at a breakpoint.
LNK$	Holds status of the Link at a breakpoint.
MSK$	Contains the search mask, initialized at 777777.
LO$	Contains the address of the lower limit of the search operation.
HI$	Contains the address of the upper limit of the search operation.
PA$	Contains the address of the first position available for inserting patches. (Note that the initial contents of LO$ show the last available position plus one.)
AX$	Contains the number of the auto-index register to be used by the breakpoint routines, initialized to 17.
RF$	Contains the current relocation factor.
SA$	Contains the normal starting address used by the apostrophe routine.
Bn$	Contains the address of breakpoint n.

These words are stored sequentially as listed; the line feed may be used to step through them.

In the following example, the mask is set to examine instruction code bits (0-3) within the limits specified by LO$ and HI$.

MSK$/	LAW 17777	740000 ↓
LO$/	CAL+11075	BEGIN-1 ↓
HI$/	END+67	END+1 ↳

After the mask and search limits have been set, the user may execute the search operation for the desired instruction class (all JMP instructions) by typing:

> JMP ⌐ EQ$ ↳

2.8   SYMBOL DEFINITIONS

If the user finds, while debugging, that more symbols would be useful he can easily define them with the following DDT-9 procedure:

S)   DDT-9 assigns the symbol S to the current location.

k(S)   DDT-9 assigns the symbol S to the location specified by the address k.

Example:

13627 (LOCAT)

Space is provided for approximately 25 additional symbols; the exact number will depend on the length of the symbols entered. If an attempt is made to enter symbols beyond the allowable limit, DDT-9 types the message OVERFLOW.

2.9   PATCH FILE OUTPUT

When the process of debugging extends to a number of sessions at the computer, it is convenient to be able to save those changes already checked out for use at later sessions. The commands described below control the output of a patch file onto paper tape.

PFO$ DDT-9 outputs all registers within the limits set by LO$ and HI$ onto the patch file. PFO$ may be given as many times as desired.

k ⌴ PFO$ Put location k only onto the patch file.

SNS$ DDT-9 puts all symbols defined during debugging onto the patch file, thus saving them for reference at later sessions.

PFE$ Close the patch file.

As many files as desired may be produced by following the sequence of commands, as follows:

PFO$
.
.   (as many as desired)
.
PFO$

SNS$ (optional)

PFE$

2.10   PATCH FILE INPUT

Because of the patch file's format, it may be loaded only by DDT-9. This is done after the user's program has been loaded in the usual manner. If a read error occurs, DDT-9 stops reading and types the message ERROR followed by a right angle bracket (>). Data up to the point of error is correctly in memory.

PFI$ DDT-9 reads in the patch file.

Typing PFI$ at this point (without repositioning the tape), will cause patch loading to continue with the patch word after the word causing the error.

Repositioning the tape by moving the tape back one block will cause PFI$ to attempt to re-read the error word. (See appendix 3 for format of the patch file.)

2.11   CO-RESIDENT SUBROUTINES

Since identical symbols may be used in two or more separately assembled or compiled, re-locatable program segments that are loaded and run together, the user must be able to specify which set of symbols DDT-9 is to use. DDT-9 initially assumes that the symbol table associated with the first program loaded (i.e., the main program) will be used. The relocation factor used by DDT-9 comes from the symbol table and is, also, initially assumed to be that of the main program. The following DDT-9 command changes both the symbol table search and the relocation factor to the named subroutine.

k HDR$ Sets DDT-9 to refer to that portion of the symbol table associated with the subroutine name k, and to use the relocation factor for that subroutine. (The memory map output by the loader shows all relocation factors.) Symbol tables are not loaded for IOPS and FORTRAN library subroutines.

HDR$ If no program name is specified, DDT-9 is reset to the initial condition, with main program symbol table and relocation factor assumed.

2.12   INDIRECT ADDRESS REFERENCES

External global symbols (those used within the program segment, but defined outside of it) are treated differently in the symbol table than those defined within the program segment. These symbols refer to a transfer vector pointing to the named register, not to the named register itself. Example:

LAB/   007603

7603 is the actual address of the storage word named LAB. This address must be used when any reference is made to LAB.

In FORTRAN programs, this condition also applies to symbols defined in DIMENSION statements.

2.13   MISCELLANEOUS FEATURES

Q$ Q$ represents the content of the currently open storage word. It makes it possible to make small changes without typing the entire contents. In the following example, Q$ represents JMP LOC+3.

LOC/   JMP LOC+3   Q$+4)
LOC/   JMP LOC+7

The period, typed alone, represents the address of the currently open or the most recently opened word.

     LOC/   JMP LOC+3   JMP .+7 ⤶

     ./   JMP LOC+7

&   The ampersand causes DDT-9 to bypass the mnemonic instruction lookup. It is necessary if the user has used a recognized mnemonic operator as a symbolic address.

     JMP/   Is invalid, but

     &JMP/   will open the word named JMP.

     LOC/   JMP GO   JMP JMP ⤶   The second JMP, in this case, is interpreted as an address.

k#   DDT-9 executes the instruction k. The AC and Link are restored to their condition before the breakpoint (if one is in effect). If the instruction is not a JMP, control returns to DDT-9, and the new AC and Link (if affected) are stored. For example,

     JMS ↵ SUBA#

will cause subroutine SUBA to be executed. SUBA cannot look for subsequent arguments. Skip instructions cause the return pointer to be incremented by one.

↑U   If the user makes a typing error, he can cancel the current line by typing control U. DDT-9 types @ as evidence of acceptance. Single character deletion (RUB-OUT) is not allowed by DDT-9. If a RUB-OUT is typed, it will be treated as a control U.

↑T   The user may interrupt his program (or DDT) at any time he desires, by typing control T. DDT-9 then types:

     DDT

     0   C(PC)   C(AC)   S(L)

     >

and waits for a command from the teletype.

APPENDIX 1
SUMMARY OF COMMANDS

Linkage Characters

+	Arithmetic plus
—	Arithmetic minus
(space)	Field separator

Breakpoints

k n"	Insert breakpoint at location k, assign number n (1-4)
n"	Remove breakpoint number n
"	Remove all existing breakpoints
!	Restart from breakpoint
n!	Restart from breakpoint, wait n times before reentering breakpoint
↑T	Restart DDT-9

Examination and Modification

k/	Open location k
↵	(Carriage return) Close the location
↓	(Line feed) Close the location, open next location
↑	(up arrow) Close the location, open the preceding location.
↑Z	(Control Z) Close the location, open addressed location, continue original sequence
↑A	(Control A) Close the location, open addressed location, start new sequence
↑X	(Control X) Close the location, open the location addressed by 15-bit transfer vector, start new sequence

Type-out Modes

NUM$	Type contents as 6-digit octal numbers
TV$	Type contents as transfer vectors
SYM$	Type contents as symbolic instructions (assumed by default)
:	Retype in alternate mode (NUM$, SYM$)
=	Retype as transfer vector
REL$	Type addresses as relative to defined symbols (assumed by default)

Type-out Modes (continued)

RLC$ Type address as relocatable numbers

ABS$ Type addresses as absolute numbers

Starts and Restarts

'   Starts user's program at normal starting point

k'   Starts user's program at location k

!   Restarts user's program from breakpoint

n!   Restarts user's program from breakpoint, waits n times before reentering breakpoint

↑T   Restart DDT-9

Searching Operations

k ⌴ EQ$ Search for words equal to k

k ⌴ UN$ Search for words not equal to k

k ⌴ ADR$ Search for instructions with effective address equal to k

Special DDT-9 Locations

AC$ Holds AC at a breakpoint

LNK$ Status of Link at a breakpoint

MSK$ Contains search mask

LO$ Lower limit of search

HI$ Upper limit of search

PA$ First unused location in patch area

AX$ Number of auto-index used by breakpoints

RF$ Current relocation factor

SA$ Normal starting address

Bn$ Address of breakpoint n

Symbol Definition

s)   Assign symbol s to the current location

k(s)   Assign symbol s to location k

Patch File Output

PFO$ Patch file output

k⌐PFO$ Single location patch file output

SNS$ Save new symbols

PFE$ Close patch file output

Patch File Input

PFI$ Read patch file

Coresident Subroutines

k HDR$ Use symbol table and relocation factor of subroutine k

HDR$ Use symbol table and relocation factor of main program

Miscellaneous Features

Q$ Contents of currently open location

                    Address of currently open or most recently opened location

&   Bypass mnemonic instruction lookup

k#   Execute the instruction k

↑U   Cancel the line

↑T   Restart DDT-9

APPENDIX 2
MNEMONIC INSTRUCTION TABLE

Memory Reference			Operate	
CAL	000000		NOP	740000*
DAC	040000		OPR	740000
JMS	100000		CMA	740001
DZM	140000		CML	740002
LAC	200000		RAL	740010
XOR	240000		RAR	740020
ADD	300000		SMA	740100
TAD	340000		SZA	740200
XCT	400000		SNL	740400
ISZ	440000		SKP	741000
AND	500000		SPA	741100
SAD	540000		SNA	741200
JMP	600000		SZL	741010
			RTL	742010
EAE Group			RTR	742020
EAE	640000		CLL	744000
			STL	744002
Input/Output			RCL	744010
IOT	700000		RCR	744020
			CLA	750000
			CLC	750001
			GLK	750010
			LAW	760000

*DDT-9 interprets 740000 as NOP.

APPENDIX 3

PATCH FILE FORMAT

DDT-9 punches the patch file in four-word blocks, including the two-word block header used by the IOPs system, with blank tape showing between the blocks. Each block carries the address and the contents of one memory word. (See figure A3-1.) The Save New Symbols command (SNS$) punches the additional symbol table area in the same manner. The PFE$ command punches an IOPs end-of-file block.

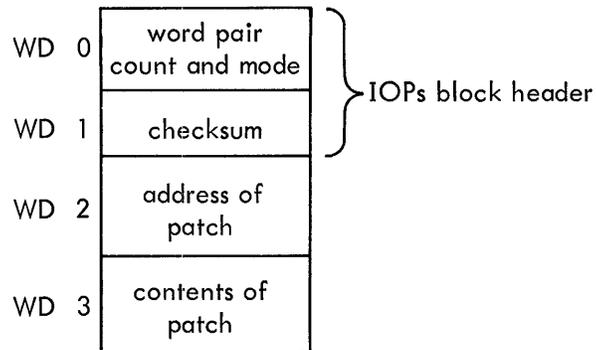Figure A3-1

EDITOR-9

ACKNOWLEDGEMENTS

The structure of the PDP-9 Text Editor control language is based in large part upon that offered by TYPSET*, a context-editing program designed and written by Jerome H. Saltzer in November, 1964, under the auspices of Project MAC, The Massachusetts Institute of Technology.

PDP-9 TEXT EDITOR

CONTENTS

CONTENTS (continued)

APPENDIX

ILLUSTRATIONS

TABLES

1   <u>INTRODUCTION</u>

The PDP-9 Text Editor (EDIT 9) is a powerful context-editing program that allows the modification and creation of symbolic source programs and other ASCII text material.* By means of commands issued from the Teletype, the Editor is directed to bring a line, or group of lines, from the input file to an internal buffer. The user may then, by means of additional commands, examine, delete, and change the contents of the buffer, and insert new text at any point in the buffer. When the line, or block of lines, has been edited, it is written into a new file on the output device.

The Editor is most frequently used to modify MACRO-9 and FORTRAN IV source programs, but it may also be used to edit any symbolic text.

The Editor operates in the ADVANCED Software System with either the I/O or Keyboard Monitor and may be used with all standard peripheral devices. The program resides in locations 13000_8 to 17763_8 of the highest memory bank present, occupying 2471_{10} registers.** Additional memory, excepting that reserved for the Monitor and the required device handlers, is utilized for block mode buffers.

*The Editor reads and writes standard IOPS ASCII lines. The characteristics of IOPS ASCII text are described in the Monitor manual (DEC-9A-MAB0-D).

**Attention is drawn to the Monitor manual, Chapter 3, for a detailed discussion of loading and memory-allocation schemes for system programs.

2 FUNCTIONAL DESCRIPTION

2.1 Control Modes

The PDP-9 Editor operates in one of two control modes; in edit (or command) mode the program accepts and acts upon control word and data strings to open and close files, to bring lines of text from an open file into the work area, to change, delete, or replace the line currently in the work area, and to insert single or multiple lines after the line in the work area. In input (or text) mode, lines from the Teletype are interpreted as text to be added to the open file. Commands are available for conveniently changing control mode.

2.2 Data Modes

Data from the input file is made available for editing in two ways: in line-by-line mode or in block mode.

2.2.1 Line-by-Line

In line-by-line data mode a single line is the unit of the input file available to the user for modification at any point. The line currently available is specified by a pointer which can be thought of as moving sequentially through the file, starting at the first line, in response to typed editing commands. When a file is opened at the beginning of an editing session, the first line of that file is brought into the work area and is available for modification. This line remains in the work area until the user requests that a new line be brought in. The pointer then moves down the file until the line requested is encountered. That line is brought to the work area and, as the "current line," can be modified. Lines previously skipped over are no longer available for editing by the user, but are written in the output file. Thus at any point in a single edit run in line-by-line mode, the user is able to modify only the portion of the input file consisting of the current line and all lines between the current line and the end of the file (i.e., the current line and all lines below it).

2.2.2 Block Data Mode

In block data mode, a user-specified portion of the input file is held in a core buffer for editing until the user requests that the contents of the buffer be added to the output file. All of the Editor commands used in line-by-line editing are employed when editing blocks; in addition, a group of commands is available for use in block mode only (see Chapter 5).

When the user is operating in block mode, commands to the Editor are honored only with respect to that portion of the input file currently occupying the buffer. The lines of text in the buffer are made available for modification through the use of normal locative requests and, moreover, may be reaccessed until the buffer is emptied by the user.

Unless deleted, lines passed over in block mode are not lost to the user (as in line-by-line mode) until the contents of the buffer are written in the output file. Consider, for example, the editing request to search for and bring in a specified line. In line-by-line mode, the result is a scan of (possibly) the entire file below the pointer. The same request in block mode provides a search of the entire buffer below the pointer, but no further.

Block mode has another advantage: rapid correction of editing command errors. If the user finds that he has typed the wrong command, he can immediately correct it, since the buffer has not been added to the output file. In line-by-line mode, a command error may cause the program to bypass a line in which a change is needed. The user would then have to punch a new input file and begin editing (more carefully) again.
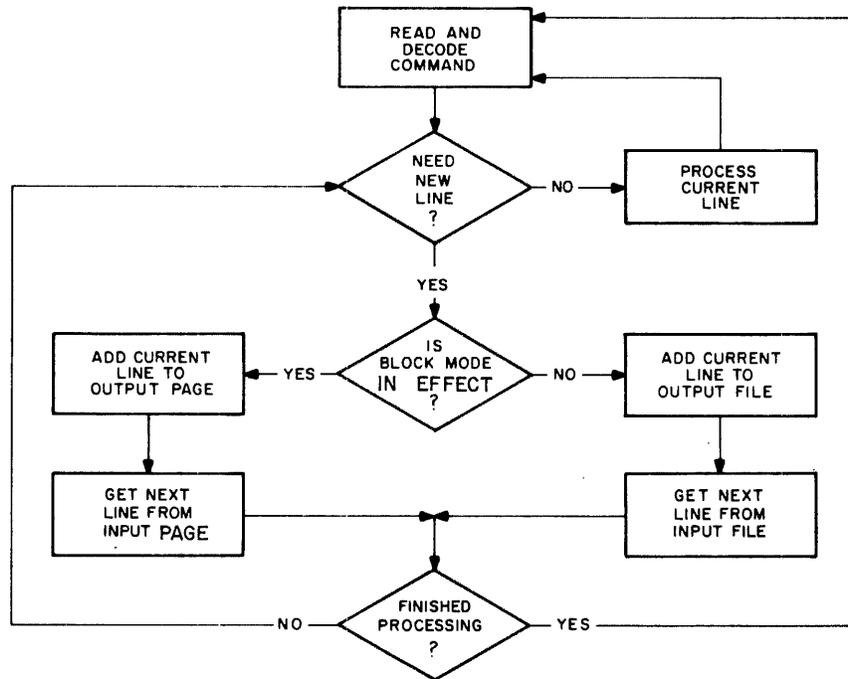
Figure 2-1   Schematic of Line Processing in
Block and Normal Modes

2.3   Data Files

2.3.1   Using Monitor I/O

The Editor makes use of the Monitor Input/Output Programming System for I/O transfers and communicates with IOPS by way of entries in the Device Assignment Table. Entries in DAT which are required by the Editor are given in Table 2-1. Methods of modifying DAT are described in the Monitor manual (DEC-9A-MAB0-D).

Table 2-1   Standard DAT Assignments for PDP-9 Text Editor

DAT Entry Number	Used For
-3	Teleprinter output; messages to user
-2	Keyboard input; text and commands
-14	File input
-15	Scratch or edit file output*
-10	Subsidiary file input

*The use of the scratch device is described in Section 2.3.3.

2.3.2   Input and Subsidiary Files

The Editor will accept file input from a maximum of two devices in addition to input from the keyboard. The first device normally holds a previously prepared file upon which changes are to be carried out. The second, the subsidiary file device, is usually the medium through which additional, previously prepared, text is inserted in the object file. Either one, or both, of these devices may be ignored by the user, in which case the Editor assumes that all data will come from the keyboard.

Care must be taken in the specification of the subsidiary input device to ensure that the data of interest residing thereon was recorded in nonfile-structured fashion. For the paper tape and card readers, this is the only recording mode. For other devices (e.g., DECtape), however, the user has the choice of writing data in either a file-structured or a nonfile-structured manner.* When the Editor is first loaded, the characteristics of the subsidiary input device are determined. If that device can be file-structured, the comment

SECONDARY INPUT DEVICE IS FILE-ORIENTED

is printed on the Teletype. The intent here is to warn the user that disaster will result if the data to be read from the device is file-structured. Note, however, that if the data to be read was recorded in

*For a discussion of data-handling conventions in file-structured and nonfile-structured input/output modes, see the Monitor manual (DEC-9A-MB00-D).

nonfile-structured fashion, then the requested device is a legal one for secondary input. Accordingly, the Editor then asks the question,

<p align="center">DO YOU WISH TO CONTINUE?</p>

The user's answer to this question is taken to indicate the nature of the data on the secondary input device. If the user's response is

<p align="center">YES)</p>

then the program will read data from the device in the normal (nonfile-structured) way. If the user's answer is NO (or anything except YES) file-structured data is assumed and return is made (via .EXIT) to the Monitor.

2.3.3 Output Files

Immediately upon receiving control after having been loaded, the Editor attempts to determine whether or not the input and scratch devices are file structured. If either one of the devices is not file structured, then the scratch device (DAT entry -15) is assigned as the final output device. If both devices are file structured, the scratch device is assigned an intermediary function and the input device is used as the final output device.

The intent, in all cases, is to allow replacement of the input file by the edited output file. This is possible only when the input and output devices can be both read and written. If replacement can be accomplished (both devices are file structured), the following sequence of events takes place when the files are closed after editing.

1. The intermediate output file is read from the scratch device and written on the input device under a temporary name.

2. The old input file is deleted from the input device.

3. The intermediate output file is deleted from the scratch device.

4. The intermediate output file, temporarily named and now residing on the input device, is given the name previously assigned to the old (now deleted) input file.

5. The output file is closed and immediately becomes available for use.

If no replacement can be accomplished, no change is ever made to the input file. If the output device is file oriented, the new edited file is properly entered in the file directory for that device under the name given in the OPEN or CLOSE command sequences.

The possible destinations of the new edited file are summarized in Table 2-2.

Note that in the process of file housekeeping, there is always at least one copy of the output file available on one, or both, of the devices. Further, the original input file is not deleted until the new file has been successfully written and closed. A system failure, therefore, can never result in total loss of data. Recovery procedures to be used in case of difficulty are outlined in Chapter 5.

Table 2-2   Output File Conventions for PDP-9 Text Editor

Input Device	Scratch or Output Device	Edited File appears on:	Input File is:
File oriented	File oriented	Input Device	Deleted
File oriented	Nonfile oriented	Output Device	Unchanged
Nonfile oriented	File oriented	Output Device	Unchanged
Nonfile oriented	Nonfile oriented	Output Device	Unchanged

2.4   Using the Break (CNTRL P) Character

Frequently, the user, having made a mistake in his command line, wishes to stop processing and reissue his request. The user, for example, may have asked erroneously for a line which is absent from the input file. When the Editor begins its search for the requested line, it will not give up until that line is found, or until the end of the input file is encountered. The user, meanwhile, has noticed his typing mistake. If control could somehow be transferred from the command processor to the command decoder, the user's temper and time might be saved.

The Editor's break, or quit, character provides the mechanism for the orderly accomplishment of the transfer. When the user types the quit character (CNTRL P) during command processing, the normal instruction sequence is interrupted when processing of the current line has been completed, edit mode is reentered, and the program reads a new edit command from the keyboard. Nothing is lost from the output file. Depending upon the command being serviced when CNTRL P was typed, the pointer is left in one of two positions. In the first (usual) case, the pointer indicates the line which was being processed when the break character appeared. This line is now the current line, and may be dealt with in the normal way. In the second case, the pointer is left between two lines. The current-line area is empty, and some locative request (e.g., NEXT) must be issued to move a line into the work area.

The break character results in program restart when the Editor is waiting for a command. In input mode, the break character results in a control mode change.

2.5   Using the Erase and Kill Characters

The Monitor allows the use of two keyboard characters for correction of the line currently being typed by the user. The Rubout key (Erase character) results in the deletion or the immediately preceding character. The Monitor echoes a back slash (\) for each Rubout typed. CNTRL U (Kill character) results in the deletion or the entire line typed so far. The Monitor echoes a commercial at sign (@) for each CNTRL U typed.

CNTRL U has a second function when used during output from the Editor to the Teletype. When the user types CNTRL U while a line is being printed, output is immediately terminated and a

carriage return is issued. CNTRL U functions in this case as the user's means of overriding his previous request for the output of tediously long lines.

3   EDITING OPERATIONS

The Editor always begins in edit mode and assumes that the user wishes to modify some (named or unnamed) file. When first loaded, or when restarted for a new file, the program types

> EDITOR
> >

on the teleprinter and waits for the user's first command.

3.1   Modifying an Existing File

If the input device is file structured (disk, drum, magnetic tape, or DECtape), the first command to the Editor must be

> OPEN filename ext↙

where "filename" is the primary name of the wanted file residing on the input device and "ext" is its extension. "Ext" may be omitted and, if so, is assumed to be SRC. If the file specified is not found in the directory, the program assumes that the user wishes to create a file named "filename ext." Accordingly, when it has been determined that the named file is absent from the input device, the Editor types

> FILE filename ext NOT FOUND.
> INPUT

Input mode is entered and subsequent lines from the Teletype are inserted in a new, temporarily named, file on the output device.

If the specified file is present on the input device, an intermediate, temporarily named, file is opened for writing on the output device and the input file is opened for reading. The user may then proceed to make the necessary changes in the input file.

If the input device is not file structured (e.g., paper tape reader, card reader), the user's first command after program initialization may be any edit request. The OPEN command is not required for nonfile structured devices.

3.2   Creating a New File

When the user wishes to create a new file, he need only issue a carriage return, thereby entering input mode. If the output device is file structured, a temporarily named file is opened for writing and text lines from the Teletype are added to it as they appear. If the output device is not file structured, the file-naming conventions are bypassed.

Where both input and output devices are file structured, the user may issue the OPEN command followed by the name he wishes to assign to his new file. Since a file of the name given is

guaranteed not to be found (if the user has properly chosen his new name), input mode will immediately be entered following the standard error message. The name specified will be assigned to the final output file if no other name is given in the CLOSE command.

3.3 Input/Edit Modes

To enter text from the Teletype, the Editor must be in input mode. To carry out an edit function on the current line, the Editor must be in edit mode.

Control mode may be changed at any time by typing a line of zero length (a line consisting of a carriage return only). The Editor command INSERT (without arguments) also causes a mode change. After the user changes control modes, the Editor types INPUT or EDIT, indicating the control mode in effect.

3.4 Block Mode

The Editor recognizes several commands which are designed to be useful in the block or page mode. In block mode, a user-specified portion of the input file is held in a core buffer until the user indicates his satisfaction with the current state of that portion. Block mode is entered via the control word BLOCK, followed by the parameter ON. When in block mode, the user may take advantage of all the locative and manipulative commands (FIND, LOCATE, CHANGE, etc.) and, in addition, may employ the MOVE command to rearrange arbitrarily long blocks of text within the buffer.

Line-by-line mode is reentered by use of the BLOCK OFF command.

3.5 Closing the New File

When the user, after modifying his input file, is satisfied that all needed changes have been carried out, he is required to close out the input and output files. The edit command

CLOSE filename ext ⤶

will initiate the sequence of events described above (Section 2.3.3).

Neither "filename" nor "ext" need be specified if previously given in the OPEN command. If "filename" and "ext" are present in the command string, they override the names given in the OPEN command.

Both "filename" and "ext" are ignored if the output device is nonfile oriented.

3.6 Error-Handling Conventions

3.6.1 Command String Errors

All mistakes in the use of edit-mode control words result in a common complaint by the Editor. Although the possible errors in usage fall into a number of distinct categories, the program

makes no attempt to differentiate among error types. The reasons for this common treatment lie in the requirement that the Editor take some cognizance of its memory allocation (relatively obscure error types need as much memory for recognition and response as do the more usual mistakes) and in the fact that the treatment rendered makes the error self-explanatory, in most cases, with respect to the difficulty encountered.

Command string errors, then, all result in the single typed comment,

NOT A REQUEST:

followed, on the next line, by the request line with which the Editor had trouble.

Usual types of command string errors include the following:

A. The edit control word issued was not among those in the program's repertoire.

B. A SIZE command was issued with a missing argument or an argument of "1."

C. A MOVE request was issued when BLOCK mode was OFF.

When BRIEF mode is ON, the Editor comment and the command line in error are replaced by a single typed question mark, thus:

? ▶

3.6.2   Premature End-of-File

During the processing of some commands, it occasionally happens that a read is attempted which moves the pointer below the last line of a logical (or physical) group. Consider, for example, the effect of a numeric argument in the GET n command line. The program reads successive lines from the subsidiary input device until exactly n lines have been read. If, in the process of reading, it is discovered that fewer than n lines are physically present on the secondary input medium (paper tape, say), then a premature end-of-file condition is said to exist. An improperly-formulated FIND request (the character string typed is absent from the file) will result in a similar condition.

Depending upon the character of the incoming group of lines (block buffer, secondary input medium, or input file), the appearance of an unexpected end-of-file causes a comment to be typed informing the user of the difficulty. The form of the message is:

$$\text{END OF} \begin{Bmatrix} \text{BUFFER} \\ \text{MEDIUM} \\ \text{FILE} \end{Bmatrix} \text{REACHED BY:}$$

followed, on the next line, by the edit request which caused the problem.

A premature end-of-file causes the pointer to be left <u>below</u> the final line of the group being read.

3.6.3   Read Errors and Line Overflow

The Editor recognizes two sorts of errors which may occur during the processing of the input file. Both errors result in an appropriate printed comment and immediate transfer of control to the command decoder. The line in error is printed and left in the work area for modification by the user.

The first type of error occurs when the input file device handler detects either incorrect parity or a faulty checksum in the incoming line. The printed comment is:

READ ERROR:

followed by the line in which the error was encountered.

The second difficulty results from the appearance of a line which is too long to be contained in the program's internal buffers. Any line of more than 90_{10} characters (not including terminator) results in the comment:

TRUNCATED:

followed by the first (leftmost) 90 characters of the long line. The remaining right-end characters are discarded.

The user has the choice, after either type of error, of modifying the line which caused the complaint (via any manipulative request) or of allowing the line to stand as is in the output file (via any locative request).

3.6.4   Block-Mode Buffer Overflow

When block mode is in effect, it is possible for an attempted addition of a line to the block-mode buffer to exceed the buffer's capacity. This might occur, for example, during the processing of a READ request if the buffer length (previously defined by a SIZE command) is too great to be accommodated by the memory available. When the capacity of the buffer is exceeded, the program types the comment:

BUFFER CAPACITY EXCEEDED BY:

followed by the line which caused the overflow. This line remains in the current-line area and the program reads a new command from the keyboard.

3.6.5   File-Naming and Calling Errors

Errors in file-name usage can be classified in three general groups. Either (1) the named file cannot be found, or (2) a name has not been given to the file at a point where one is needed, or (3) a name has been given which cannot be used.

3.6.5.1   Absent File - If the file named in the OPEN request line cannot be found on the device associated with DAT slot-14, the assumption is made that the user wishes to create a new file with the

the name given. The program prints the comment:

FILE filename ext NOT FOUND.

and changes to input mode.

3.6.5.2 Absent File Name - If no file name is given either in an OPEN request line or as an argument
to the CLOSE command, the program, after attempting to process the CLOSE request, will print:

NO FILE NAME GIVEN.

The next edit request must be another CLOSE naming the file.

If no OPEN command is issued (a new file is being created), any locative request (FIND,
NEXT) will result in the comment:

NO INPUT FILE PRESENT.

3.6.5.3 Identically-Named Files - The problem of duplicate file names is apparent on two levels.
In the first case, it is possible for a previous edit run to have been aborted with one of the Editor's
temporary files (normally .TFIL1 EDT) closed on the output device. The closing of the temporary file
created during the current edit run will result in the deletion of the like-named file from the previous
run, perhaps to the user's keen disappointment. To enable the retrieval of prior work, the Editor types
the comments:

FILE .TFIL1 EDT IS PRESENT ON OUTPUT DEVICE.

DO YOU WISH TO DELETE IT?

If the user's response to this question is

YES 𝒍

then the version of the file on the output device is deleted and processing continues as usual. If the
user's response is

NO (or anything except YES)

then return is made (via .EXIT) to the Monitor. The user may then rename .TFIL1 EDT.

At the second level, it may happen that the file name given in a CLOSE sequence is identical
to that of another file on the (current) output device. In this case, the program types:

PLEASE USE ANOTHER NAME.

A second CLOSE request (with a unique name) may then be issued.

4   EDITOR COMMANDS

When edit mode is in effect, the following commands result in the specified activity. Abbreviations for most commands consist of the initial characters of those commands. Legal abbreviations are given in square brackets. Optional arguments are given in parentheses.

Certain commands (e.g., FIND, RETYPE) require the presence of arguments. Others (DELETE, NEXT) may take explicit arguments at the option of the user. All commands must be separated from their argument strings by a single blank character. This blank delimiter is considered by the Editor to be a part of the command itself, not part of the argument string which follows the command. Thus, the command

    RETYPE ⊔ /COMMENT ⟩

results in the following line:

    /COMMENT

If more than one blank appears between the command and its argument string, all blanks except the first are taken as part of the argument. Thus,

    FIND ⊔ ⊔ ⊔ /COMMENT ⟩

results in a search for the line which begins with the character string

    ⊔ ⊔ /COMMENT

4.1   OPEN (filename (ext)) ⟩

The file whose name is "filename" and whose extension is "ext" is searched for on the input device. If a file of this name is not found, a message is printed on the Teletype and the mode is changed to input. An intermediate write file is opened on the output device and lines from the keyboard are written into it as they are completed. "Ext," if not given, is assumed to be SRC.

If the file specified is found on the input device, it is opened for reading. Subsequent typed lines are interpreted as Editor commands.

Neither file name nor extension need be given if the input device is nonfile oriented.

4.2   CLOSE (filename (ext)) ⟩

If an input file is present, all lines in that file falling below the current line are appended to the output file and the output file is closed. If no input file is present, the current line is added to the output file and the output file is closed. No further editing is permitted.

If the extension is omitted, and none was assigned in the OPEN command line, the extension is assumed to be SRC. If no file name is given, the name assigned in the OPEN command line is used.

Neither "filename" nor "ext" need be given for nonfile-oriented output devices.

4.3   NEXT [N] (⊔n)⟩

The pointer is moved past the next n lines, beginning with the line currently in the work area. Line n + 1 is brought into the work area for modification. Lines skipped over are added to the output file. If omitted, n is assumed to be 1. If the command results in the pointer moving past the last line of the file (or buffer, if block mode is on) the error message

$$\text{END OF} \left\{ \begin{array}{l} \text{FILE} \\ \text{BUFFER} \end{array} \right\} \quad \text{REACHED BY:}$$

NEXT n

is printed.

4.4   PRINT [P] (⊔n)⟩

n lines from the input file (or buffer, in block mode), including the current line, are printed on the Teletype. The pointer is left at the last line printed; n is assumed to be 1 if omitted.

If, as a result of the command, the pointer moves past the last line of the file, the error message

$$\text{END OF} \left\{ \begin{array}{l} \text{FILE} \\ \text{BUFFER} \end{array} \right\} \quad \text{REACHED BY:}$$

PRINT n

is printed.

4.5   FIND [F] ⊔ string ⟩

The input file or buffer is searched, beginning with the line following the current line, for the next occurrence of a line which begins with the character group "string." If the search is success-ful, the line beginning with "string" is brought into the work area. If the search is unsuccessful (pointer moves past end of file), the end-of-file error message is printed.

"String" may contain any number of characters.

4.6   LOCATE [L] ⊔ string ⟩

The input file is searched, beginning with the line following the current line, for the next occurrence of a line which contains the character group "string". If the search is successful, the line which satisfies the search is brought to the work area. If the search is unsuccessful, the end-of-file message is printed and the pointer is moved to the top of the file.

"String" may contain any number of characters.

4.7 DELETE [D] (⊔n)

n lines, including the current line, are deleted from the input file. The line following the last line deleted becomes the current line. If n is omitted, only the current line is deleted. If n is large enough to cause the pointer to move past the end of the file, the end-of-file error message is printed.

4.8 BOTTOM [B]

The pointer is moved to the final line in the input file (or buffer) which then becomes the current line. Lines skipped over in the process of moving the pointer are added to the output file.

4.9 RETYPE [R] ⊔ line

The character string "line" replaces the current line. The new line is left in the work area and may be subsequently modified.

4.10 INSERT [I] ⊔ line

The current line is added to the output file and the character string "line" is taken as the current line. Note that insertions are always made below the current line. The program remains in edit mode when command processing is completed.

4.11 INSERT [I]

The current line is added to the output file and the mode is changed from edit to input. Subsequent lines are interpreted as text to be added to the output file.

4.12 GET [G] (⊔n)

n lines from the subsidiary input device are added to the output file. New lines are added below the current line. When command processing is complete, the nth line read is left in the work area as the current line. If n is omitted, it is assumed to be 1.

If an end-of-medium condition is encountered on the subsidiary input device before n lines are read, the error message
                    END OF MEDIUM REACHED BY:
                    GET n
is printed. The pointer remains at the last line read.

4.13   CHANGE [C] ⊔ q string1q string2q ⟩
_____

In the current line, the first character group ("string1") which matches that occurring between the first pair of quote characters (q's, in this case) is replaced by the character group ("string2") appearing between the second pair of quote characters. The quote character chosen by the user may be any graphic (including blank) which does not appear in either of the character strings quoted. Both "string1" and "string2" may contain any number of characters, including zero. If verify mode is in effect, the program will print the new current line on the Teletype when the requested change has been accomplished. Examples of change requests:

Current line:	NXTLIN		JMS TYPOUT	/PRNT THE LINE.

a.   In the comment, spell "PRINT" properly.

Request:	CHANGE⊔/RN/RIN/⟩			
New line:	NXTLIN		JMS TYPOUT	/PRINT THE LINE.

b.   Make the "JMS" a "JMP*".

Request:	CHANGE⊔XSXP*X⟩			
New line:	NXTLIN		JMP* TYPOUT	/PRINT THE LINE.

c.   Delete the "t" in the tag.

Request:	C⊔/T//⟩			
New line:	NXLIN		JMP* TYPOUT	/PRINT THE LINE.

4.14   TOP [T] ⟩
_____

Move the pointer to the beginning of the edited file or buffer. The first line of the file becomes the current line.

4.15   VERIFY [V] ⊔ { ON / OFF } ⟩
_____

Set the verify mode according to the parameter. When verify mode is on, text lines are printed in response to certain editing commands, for example:

1.   The line brought into the work area as a result of a FIND or LOCATE request is printed.
2.   The last line of the file, brought in by the BOTTOM request, is printed.
3.   The new line resulting from a CHANGE request is printed.

When verify mode is off, only error messages are printed. After the Editor is loaded initially, verify mode is on.

The command

VERIFY [V] ⟩

(without arguments) is equivalent to

VERIFY [V] ⊔ON⟩

4.16   OVERLAY [O](⊔n) ⟩

Starting with the current line, n lines (or the current line only, if n is omitted) are deleted from the input file. Control mode is changed to input with the normal typed program response,

INPUT

Subsequent typed lines are interpreted as text intended to replace the lines so OVERLAYed.

4.17   APPEND [A] ⊔ string ⟩

"String" is added to the current line following the last data character and preceding the terminating carriage return. Thus, to add a comment to the current line

JMS GETNUM

the command might be

APPEND ⊔ →/GET DECIMAL ARGUMENT.⟩

The new current line would be

JMS GETNUM →/GET DECIMAL ARGUMENT.

If "string" is absent, the current line is unchanged.

4.18   BRIEF⊔ {ON OFF ⟩

Set brief mode according to the ON/OFF parameter. Brief mode results in the abbreviated printing of the current line during the servicing of some commands. An attempt is made to print only the tag, operation code, and address fields of lines brought in as a result of the FIND, LOCATE, and BOTTOM commands. In addition, the printing of the new line resulting from a CHANGE request is terminated at the last newly-inserted character.

Brief mode is set to off initially. The setting of the brief mode indicator is of no consequence when verify mode is off.

The command

BRIEF⟩

(without arguments) is equivalent to

BRIEF⊔ON⟩

4.19   BLOCK⊔{ON OFF}↓

Set block mode according to the parameter. When block mode is on, the editing commands READ, WRITE, and MOVE are accepted by the program; these commands are treated as illegal if block mode is off. When block mode is in effect, the program treats several lines as a subfile, retaining them internally in a block buffer. In block mode, editing commands which move the pointer reference only those lines currently residing in the buffer. The contents of the buffer are saved until a WRITE command is encountered or until, by way of the DELETE command, it is emptied.

When block mode is off, sequential lines in the input file are moved singly to the work area and are not available for reexamination after the pointer has been moved to a later line.

When the Editor is initially loaded, block mode is set to on if either the input or the scratch device is nonfile oriented. If both devices are file oriented, block mode is set off.

The command
        BLOCK↓
(without arguments) is equivalent to
        BLOCK⊔ON↓

4.20   SIZE [S] ⊔ n ↓

Set the total number of lines which will occupy a buffer (in block mode) to n. The SIZE command may be issued at any time, and takes effect when the next group of lines is inserted in the buffer via a READ command. n is initially set to 55_{10}. n must be greater than 1.

NOTE

Commands 4.21–4.23 are legal only in the BLOCK data mode.

4.21   READ ↓

Read sequential lines from the input file, inserting them in the buffer as they are encountered, until the number of lines in the buffer is equal to the argument specified in the SIZE request. The pointer is set to the first line of the buffer when the operation is complete.

The READ request will not be accepted if any lines remain in the current buffer. The buffer must have been cleared by DELETE requests or a WRITE command.

The READ request is treated as illegal if block mode is off.

4.22   WRITE ↓

Add the current contents of the block buffer to the output file and clear the buffer. Nothing is output if the buffer is empty. This request is illegal if block mode is not in effect.

4.23 MOVE⎵TAG1 $(\pm n_1)$ ⎵ TAG2$(\pm n_2)$ ⎵ TAG3 $(\pm n_3)$ ⟩

Perform a block transfer of several lines in the buffer. The inclusive limits of the block to be moved are defined by the first two arguments (TAG1 and TAG2). The destination of the block so transferred is defined by the third argument (TAG3).

TAG1, TAG2, and TAG3 are symbolic labels in lines anywhere in the the buffer. The n_i are optional augments to be used when block-limiting lines are not labeled.

At the completion of command processing, the block of lines between and including those labeled TAG1 and TAG2 (augmented, if desired) are repositioned to appear after the line labeled TAG3. The pointer is left at the top of the buffer.

This command is legal only in block mode.

4.24 EXIT

Control is transferred from the Editor to the Monitor. This command is illegal if any file is open for reading or writing when it is issued, i.e., it may only be given as the first command after Editor initialization and the message

EDITOR
>

5   RECOVERY PROCEDURES

In case of a hardware or system failure, the user may recover at the point at which the last complete version of the edited output file was closed. The Editor, in preparing intermediate files, assigns them temporary names. Thus, in the event of disaster, one (or both) of the following files may be found.

.TFIL1 EDT and .TFIL2 EDT both contain the version of the edited file extant at the point at which the crash occurred. No editing is lost. If neither of these files is present, the file specified in the OPEN command contains the version of the file extant at the time the latest TOP command was issued. All editing taking place after the latest TOP command is lost. If neither .TFIL1 EDT nor .TFIL2 EDT is found and if no file name was given in the OPEN command, no recovery is possible.

6   EXAMPLES OF EDITING REQUESTS

This chapter contains illustrations of one complete iteration through the modification process using the Editor.

Figure 6-1 shows the assembly listing of a sample input file.

Figure 6-2 shows the same listing marked for correction.

Figures 6-3A and 6-3B show the hard-copy output of the editing session.   The sequence numbers at the right margin are not program generated, but were added later for reference.

Figure 6-4 is the assembly listing of the new, edited file showing the results of the editing run.

```
                                            /SUBROUTINE PACK, 7-BIT CHARS TO IOPS ASCII.
                                            /CALL:      JMS PACK
                                            /           FROM
                                            /           TO
        00000   R   000000   A       PACK       0
        00001   R   200000   R                  LAC PACK                /GET FROM ADRESS.
  U     00002   R   040044   R                  DAC PFROM               /GIVE TO FROM POINTER.
        00003   R   220000   R                  LAC* PACK               /GET ADDRESS OF TO ARRAY.
  U     00004   R   040052   R                  DAC PTO                 /GIVE TO OUTPUT POINTER.
  U     00005   R   040047   R                  DAC PLRH                /SAVE AS START ADDRESS.
        00006   R   440000   R                  ISZ PACK                /BUMP TO RETURN.
        00007   R   777773   A       PLOOP1     LAW -17773              /SET UP
  U     00010   R   040045   R                  DAC PK5CHR
  U     00011   R   220044   R       PLOOP2     LAC* PFROM              /GET NEXT WORD IN INPUT ARRAY.
  LU    00012   R   000000   R                  SAC (-1                 /TERMINATOR?
        00013   R   741000   A                  SKP                     /NO, SKIP.
        00014   R   600033   R                  JMP PCLOS               /YES, GO CLOSE OUTPUT ARRAY.
  U     00015   R   440044   R                  ISZ PFROM               /POINT TO NEXT WORD.
  U     00016   R   440044   R                  ISZ PFROM               /POINT TO NEXT WORD.
  U     00017   R   040055   R                  DAC PWRD3               /SET UP TO ROTATE.
  U     00020   R   100051   R       PLOOP7     JMS PRAL7
  U     00021   R   440046   R                  ISZ PK5CR               /5 CHARS IN.Q
  U     00022   R   600043   R                  JMP LPOOP2              /NO, GET ANOTHER.
  U     00023   R   200054   R                  LAC PWRD2               /WORD PAIR COMPLETE.
        00024   R   740010   A                  RAL                     /CLEAR PAIR BIT 35.
  U     00025   R   040054   R                  DAC PWRD2
  U     00026   R   200053   R                  LAC PWRD1               /GET FIRST WD OF PAR..
        00027   R   744010   A                  RAL!CLL                 /BIT 0 OF WD 2.
  U     00030   R   060052   R                  DAC* PTO                /INSERT FIRST WD IN OUT ARRAY.
  U     00031   R   440052   R                  ISZ PTO                 /BUMP OUT ADDRESS.
  U     00032   R   600050   R                  JMP PLOOP1              /GO SET UP NEXT PAIR.
        00033   R   777773   A       PCLOS      LAW 17773               /MAKE SURE PAIR IS COMPLETE.
  J     00034   R   540045   R                  SAD PK5CHR
        00035   R   600020   R                  JMP PLOOP7              /INCOMPLETE PAIR.
        00036   R   750001   A                  CLA!CMA                 /FORM WORD PAIR COUNT
  U     00037   R   340047   R                  TAD PLRH                /START ADDRESS.
        00040   R   740001   A                  CMA
  U     00041   R   340052   R                  TAD PTO                 /LESS END ADDRESS.
        00042   R   620000   R                  JMP* PACK               /RETURN TO CALLER.
                    000000   A                  .END
        00057   R   777777   A *LIT
```

Figure 6-1   Sample Input File

LEFT-ADJUSTED   NON-HEADERED

.GLOBL PACK, PRAL7, PWRD1, PWRD2, PWRD3

ON RETURN, AC HOLDS TOTAL WORDS OCCUPIED BY PACKED ARRAY. A WORD OF ALL 1'S MUST TERMINATE THE INPUT (UNPACKED) ARRAY.

```
                                    /SUBROUTINE PACK, 7-BIT CHARS TO IOPS ASCII.
                                    /CALL:    JMS PACK
                                    /         FROM    /START OF INPUT ARRAY.
                                    /         TO      /START OF OUTPUT ARRAY.
      00000  R  000000  A    PACK    0
      00001  R  200000  R            LAC*  [LAC PACK]      /GET FROM ADRESS.
U     00002  R  040044  R   (ISZ PACK→) DAC PFROM          /GIVE TO FROM POINTER.
      00003  P  220000  R            LAC* PACK             /GET ADDRESS OF TO ARRAY.
U     00004  R  040052  R            DAC PTO               /GIVE TO OUTPUT POINTER.
U     00005  R  040047  R            DAC PLRH              /SAVE AS START ADDRESS.
      00006  P  440000  R            ISZ PACK              /BUMP TO RETURN.
      00007  R  777773  A    PLOOP1  LAW 17773             /SET UP
U     00010  R  040045  R            DAC PK5CHR            /15-CHARACTER COUNTER.
U     00011  R  220044  R    PLOOP2  LAC* PFROM            /GET NEXT WORD IN INPUT ARRAY.
LU    00012  R  000000  R    (SAD)   SAD LIT (ENDCHR)      /TERMINATOR?
      00013  R  741000  A            SKP                   /NO, SKIP.
      00014  R  600033  R            JMP PCLOS             /YES, GO CLOSE OUTPUT ARRAY.
U     00015  R  440044  R            ISZ PFROM             /POINT TO NEXT WORD.
U     00016  R  440044  R            ISZ PFROM             /POINT TO NEXT WORD.
U     00017  R  040055  R            DAC PWRD3             /SET UP TO ROTATE.
U     00020  R  100051  R    PLOOP7  JMS PRAL7
U     00021  R  440046  R            ISZ PK5CHR (PK5CHR)   /5 CHARS IN?
U     00022  R  600043  R            JMP PLOOP2 (PLOOP2)   /NO, GET ANOTHER.
U     00023  R  200054  R            LAC PWRD2             /WORD PAIR COMPLETE.
      00024  R  740010  A    (RAL!CLL) RAL                 /CLEAR PAIR BIT 35.
U     00025  R  040054  R            DAC PWRD2
U     00026  R  200053  R            LAC PWRD1             /GET FIRST WD OF PAIR.
      00027  R  744010  A    (RAL)   RAL!CLL               /BIT 0 OF WD 2.
U     00030  R  060052  R            DAC* PTO              /INSERT FIRST WD IN OUT ARRAY.
U     00031  R  440052  R            ISZ PTO               /BUMP OUT ADDRESS.
U     00032  R  600050  R            JMP PLOOP1 (PLOOP1)   /GO SET UP NEXT PAIR.
      00033  R  777773  A    PCLOS   LAW 17773             /MAKE SURE PAIR IS COMPLETE.
U     00034  R  540045  R  (DZM PWRD3) SAD PK5CHR          /INCOMPLETE PAIR.
      00035  R  600020  R            JMP PLOOP7
      00036  R  750001  A (ENDCHR LAW -1) CLA!CMA          /FORM WORD PAIR COUNT.
U     00037  R  340047  R  (TAD* PACK) TAD PLRH            /START ADDRESS.
      00040  R  740001  A            CMA
U     00041  R  340052  R  (ISZ PACK→) TAD PTO            /LESS END ADDRESS.
      00042  R  620000  R            JMP* PACK             /RETURN TO CALLER.
             000000  A            .END
      00057  R  777777  A    *LIT
```

PFROM   0
PTO   0
PK5CHR   0

Figure 6-2   Input File Listing Marked for Correction

```
EDITOR                                                                  1
>OPEN PACK SRC                                                          2
>FIND /SUBROUT                                                          3
 /SUBROUTINE PACK, 7-BIT CHARS TO IOPS ASCII.                          4
>OVERLAY 1                                                              5
INPUT                                                                   6
/SUBROUTINE PACK, 7-BIT LEFT-ADJUSTED CHARS TO NON-HEADERED IOPS       7
/ASCII. ON RETURN, AC HOLDS TOTAL WORDS OCCUPIED BY PACKED ARRAY.      8
/A WORD OF ALL 1'S MUST O\TERMINATE THE INPUT (UNPACKED) ARRAY.        9
                                                                       10
                                                                       11
EDIT                                                                   12
>LOCATE FROM                                                           13
 /          FROM                                                       14
>APPEND              /START OF INPUT ARRAY.                            14
>NEXT                                                                  15
>APEND               /START OF OUTPUT ARRAY.                           16
NOT A REQUEST:                                                         17
 APEND               /START OF OUTPUT ARRAY.                           18
>APPEND              /START OF OUTPUT ARRAY.                           19
>PRINT 1                                                               20
 /          TO                   /START OF OUTPUT ARRAY.               21
>INSERT     .GLOBL PACK, PRAL7, PWRD1, PWRD2, PWRD3                    22
>L LAC                                                                 23
            LAC PACK             /GET FROM ADRESS.                     24
>CHANGE LAC/LAC*/                                                      25
            LAC PACK             /GET FROM ADRESS.                     26
CHANGE /LAC/LAC*/                                                      27
            LAC* PACK            /GET FROM ADRESS.                     28
>NEXT 1                                                                29
>INSERT                                                                30
INPUT                                                                  31
            ISZ PACK             / BUMP TO "TO" ADDRESS.               32
                                                                       33
EDIT                                                                   34
>PRINT                                                                 35
            ISZ PACK             / BUMP TO "TO" ADDRESS.               36
>BRIEF ON                                                              37
>C ./ ./.                                                              38
            ISZ PACK             /                                     39
>PRINT                                                                 40
            ISZ PACK             /BUMP TO "TO" ADDRESS.                41
>L PLBH                                                                42
            DAC PLBH                                                   43
>BRIEF OFF                                                             44
>PRINT                                                                 45
            DAC PLBH             /SAVE AS START ADDRESS.               46
>DELETE 2                                                              47
>PRINT                                                                 48
 PLOOP1     LAW 17773            /SET UP                               49
>N 1                                                                   50
>A          /5-CHARACTER COUNTER.                                      51
>L (                                                                   52
            SAC (-1              /TERMINATOR?                          53
>VERIFY OFF                                                            54
>C /SAC/SAD/                                                           55
>C /(-1/ENDCHR/                                                        56
>V ON                                                                  57
>P                                                                     58
            SAD ENDCHR                       /TERMINATOR?              59
>N                                                                     60
>D                                                                     61
>N                                                                     62
>P                                                                     63
            ISZ PFROM            /POINT TO NEXT WORD.                  64
```

Figure 6-3A   Hard-Copy Output of Editing Session

```
>D                                                                      65
>P                                                                      66
            ISZ PFROM              /POINT TO NEXT WORD.                  67
>F PL                                                                   68
 PLOOP7     JMS PRAL7                                                   69
>N                                                                      70
>RETYPE     ISZ PK5CHR            /5 CHARS IN?                          71
>N                                                                      72
>C ,LP,PL,                                                             73
            JMP PLOOP2            /NO, GET ANOTHER.                     74
>N 2                                                                    75
>CHANGE /L/L!CLL                                                       76
            RAL!CLL              /CLEAR PAIR BIT 35.                    77
>L RAL                                                                  78
            RAL!CLL              /BIT 0 OF WD 2.                        79
>C /!CLL//                                                             80
            RAL                  /BIT 0 OF WD 2.                        81
>L JMP                                                                  82
            JMP PL00P1           /GO SET UP NEXT PAIR.                  83
>CHANGE /00/00/                                                        84
            JMP PLOOP1           /GO SET UP NEXT PAIR.                  85
>N                                                                      86
>                                                                      87
INPUT                                                                   88
            DZM PWRD3                /FILL PAIR WITH ZEROES.            89
                                                                       90
EDIT                                                                    91
>L !                                                                   92
            CLA!CMA              /FORM WORD PAIR COUNT                  93
>R ENDCHR   LAW -1               /FORM WORD PAIR COUNT.                 94
>N                                                                      95
>R          TAD* PACK               /START ADDRESS.                    96
>L PTO                                                                  97
            TAD PTO              /LESS END ADDRESS.                     98
>INSERT     ISZ PACK                                                   99
>BOTTOM                                                                100
            .END                                                      101
>OVERLAY                                                              102
INPUT                                                                 103
PFROM       0                                                        104
PRO@PTO     0                                                        105
PK5CHR      0                                                        106
            .END                                                     107
                                                                     108
EDIT                                                                 109
>TOP                                                                 110
>L ADRESS                                                            111
            LAC* PACK            /GET FROM ADRESS.                    112
>C /ADR/ADDR/                                                        113
            LAC* PACK            /GET FROM ADDRESS.                   114
>LOCATE ..                                                           115
            LAC PWRD1            /GET FIRST WD OF PAR..                116
>V OFF                                                               117
>C /R./IR/                                                           118
>PRINT                                                               119
            LAC PWRD1            /GET FIRST WD OF PAIR.               120
>CLOSE                                                               121
                                                                     122
 EDITOR                                                              123
>EXIT                                                                124
                                                                     125
MONITOR                                                              126
                                                                     127
$                                           .                        128
```

Figure 6-3B   Hard-Copy Output of Editing Session (continued)

```
                                /SUBROUTINE PACK, 7-BIT LEFT-ADJUSTED CHARS TO NON-HEADERED IOPS
                                /ASCII. ON RETURN, AC HOLDS TOTAL WORDS OCCUPIED BY PACKED ARRAY.
                                /A WORD OF ALL 1'S MUST TERMINATE THE INPUT (UNPACKED) ARRAY.
                                /CALL:     JMS PACK
                                /          FROM                      /START OF INPUT ARRAY.
                                /          TO                        /START OF OUTPUT ARRAY.
                                           .GLOBL PACK, PRAL7, PWRD1, PWRD2, PWRD3
00000  R  000000  A  PACK       0
00001  R  220000  R             LAC* PACK                 /GET FROM ADDRESS.
00002  R  040042  R             DAC PFROM                 /GIVE TO FROM POINTER.
00003  R  440000  R             ISZ PACK                  /BUMP TO "TO" ADDRESS.
00004  R  220000  R             LAC* PACK                 /GET ADDRESS OF TO ARRAY.
00005  R  040043  R             DAC PTO                   /GIVE TO OUTPUT POINTER.
00006  R  777773  A  PLOOP1     LAW 17773                 /SET UP
00007  R  040044  R             DAC PK5CHR                /5-CHARACTER COUNTER.
00010  R  220042  R  PLOOP2     LAC* PFROM                /GET NEXT WORD IN INPUT ARRAY.
00011  R  540034  R             SAD ENDCHR                          /TERMINATOR?
00012  R  600030  R             JMP PCLOS                 /YES, GO CLOSE OUTPUT ARRAY.
00013  R  440042  R             ISZ PFROM                 /POINT TO NEXT WORD.
00014  R  040050  V             DAC PWRD3                 /SET UP TO ROTATE.
00015  R  100045  V  PLOOP7     JMS PRAL7
00016  R  440044  R             ISZ PK5CHR                /5 CHARS IN?
00017  R  600010  R             JMP PLOOP2                /NO, GET ANOTHER.
00020  R  200047  V             LAC PWRD2                 /WORD PAIR COMPLETE.
00021  R  744010  A             RAL!CLL                   /CLEAR PAIR BIT 35.
00022  R  040047  V             DAC PWRD2
00023  R  200046  V             LAC PWRD1                 /GET FIRST WD OF PAIR.
00024  R  740010  A             RAL                       /BIT 0 OF WD 2.
00025  R  060043  R             DAC* PTO                  /INSERT FIRST WD IN OUT ARRAY.
00026  R  440043  R             ISZ PTO                   /BUMP OUT ADDRESS.
00027  R  600006  R             JMP PLOOP1                /GO SET UP NEXT PAIR.
00030  R  777773  A  PCLOS      LAW 17773                 /MAKE SURE PAIR IS COMPLETE.
00031  R  140050  V             DZM PWRD3                 /FILL PAIR WITH ZEROES.
00032  R  540044  R             SAD PK5CHR
00033  R  600015  R             JMP PLOOP7                /INCOMPLETE PAIR.
00034  R  777777  A  ENDCHR     LAW -1                    /FORM WORD PAIR COUNT.
00035  R  360000  R             TAD* PACK                 /START ADDRESS.
00036  R  740001  A             CMA
00037  P  340043  R             TAD PTO                   /LESS END ADDRESS.
00040  R  440000  R             ISZ PACK
00041  R  620000  R             JMP* PACK                 /RETURN TO CALLER.
00042  R  000000  A  PFROM      0
00043  R  000000  A  PTO        0
00044  R  000000  A  PK5CHR     0
          000000  A             .END
```

Figure 6-4   File Resulting From Editing Session

APPENDIX 1
SUMMARY OF EDITING COMMANDS

Editor-Monitor Communication

Command	Abbreviation	Activity	Line Number*	Section
EXIT	n/a	Transfer control to Monitor.	124	4.24

File Housekeeping

Command	Abbreviation	Activity	Line Number	Section
OPEN nm ext	n/a	Prepare input file (named "nm ext") for editing.	2	4.1
CLOSE	n/a	Terminate editing on input file.	121	4.2

Locative Requests

Command	Abbreviation	Activity	Line Number	Section
FIND string	F	Bring first line beginning with "string" to work area.	3,68	4.5
LOCATE string	L	Bring first line containing "string" to work area.	12,52	4.6
NEXT	N	Bring next consecutive line to work area.	15,70	4.3
BOTTOM	B	Bring last line of file to work area.	100	4.8
TOP	T	Reset pointer to beginning of file	110	4.14
PRINT	P	Print the current line on the Teletype	20,58	4.4

Manipulative Requests

Command	Abbreviation	Activity	Line Number	Section
DELETE	D	Discard the current line.	47,61	4.7
RETYPE string	R	Replace current line with "string".	71,94	4.9
INSERT string	I	Add "string", as a complete line, to the file after (below) the current line.	99	4.10
CHANGE /string1/string2/	C	Replace, in the current line, the first occurrence of "string1" with "string2".	25,27,38	4.13
OVERLAY	O	Replace multiple lines.	5,102	4.16
APPEND string	A	Add "string" at the rightmost end of the current line.	14,16,19	4.17

*Entries under " Line Number" refer to line sequence numbers (in Figure 6-3) where examples of command usage are to be found.

Mode Control

Command	Abbreviation	Activity	Line Number	Section
VERIFY {ON OFF}	V	Set verify mode to print (ON) or ignore printing (OFF) lines after processing CHANGE, LOCATE, and FIND requests.	54,57	4.15
BLOCK {ON OFF}	n/a	Set program to operate in block mode (ON) or in line-by-line mode (OFF).		4.19
BRIEF {ON OFF}	n/a	Set brief mode to print truncated (ON) or full (OFF) lines	37,44	4.18

Input/Output Requests

Command	Abbreviation	Activity	Line Number	Section
READ	n/a	Fill block buffer from input file.		4.21
WRITE	n/a	Add block buffer to output file.		4.22
GET	G	Add lines from subsidiary input device after (below) current line.		4.12

Miscellaneous Requests

Command	Abbreviation	Activity	Line Number	Section
SIZE	S	Set total lines to occupy block buffer.		4.20
INSERT	I	Change mode to input.	30	4.11

PIP-9

CONTENTS

CONTENTS (cont)

TABLES

1.   INTRODUCTION

PIP-9 (for Peripheral Interchange Program) is a utility program in the PDP-9 ADVANCED Software System used to transfer data files from one standard peripheral device to another. PIP operates under Monitor control, using the Monitor I/O device handlers.

Files may be verified, renamed, deleted, combined or split into segments. Entire DECtapes, or individual DECtape blocks, may be copied and verified. File directories may be listed or initialized. Some of these functions and other subsidiary functions may be combined by inserting optional switches when the user types a command string to PIP-9.

The following peripheral devices may be used as either input (source) or output (destination):

	Mnemonic
DECtape (TC02 Control Unit with TU55 Transports)	DTn
Paper Tape Reader/Punch (PC02)	PR (Reader)
	PP (Punch)
Line Printer (Type 647) (output only)	LP
Teletype (KSR 33 or KSR 35)	TT
Card Reader (CR01E or CR02B)	CR

Later versions of PIP-9 will also transfer files on magnetic tape and disk.

2.   DEVICE ASSIGNMENTS

Before using PIP, the user must be sure that the peripheral devices he plans to use are assigned to positive slots in the Monitor's Device Assignment Table. This is for use by PIP. When typing his command string, the user specifies devices by writing mnemonic codes, such as DT2, PR or TT.

2.1   I/O Monitor System

In paper tape I/O Monitor systems, where the Device Assignment Table is fixed, the user need not be concerned with .DAT slot assignments. Line Printer and card reader users must be certain that the appropriate handlers are included in their systems.

2.2   Keyboard Monitor

In Keyboard Monitor systems, the user must be sure that the devices he will use are assigned .DAT slots. He should use the Monitor REQUEST PIP command to get a typeout of all current .DAT slot assignments. If a device he plans to use is not listed, he may use an ASSIGN command to assign that device to any positive .DAT slot, with the exception of .DAT slot 1 which must always be assigned to the system device. The most complete handler, (e.g. DTA, PPA, etc.), must be assigned. If the same device is to be used as both the source and destination device, it must be assigned to two .DAT slots.

1

Since these .DAT slot assignments are for use by PIP, they need not be remembered by the user. Systems distributed by DEC initially have the assignments shown below in Table 2-1.

Table 2-1
Initial .DAT Slot Assignments

.DAT Slot No.	Assignment
1	DTA0
2	DTA1
3	DTA2
4	TTA0
5	PRA0
6	PPA0
7	DTA1
10	DTA2

3.   PIP-9 COMMAND STRING: GENERAL

Once in core memory, whether in an I/O Monitor or Keyboard Monitor environment PIP-9 informs the user of its readiness to accept keyboard commands by outputting the following on the teleprinter:

PIP
>

The user may then type a command string to PIP-9 on the same line as the right angle bracket (>). Successful completion and readiness for the next command is normally acknowledged by "CR, LF, >" unless there has been intermediate output to the teleprinter by PIP. In the latter case, the initial response (PIP, CR, LF, >) is output once again for ease of later printout examination. PIP command strings are of the following general form:

$$a \quad dd \begin{bmatrix} : \\ SPACE \end{bmatrix} fname \begin{bmatrix} ; \\ SPACE \end{bmatrix} ext \ (x) \longleftarrow sd \begin{bmatrix} : \\ SPACE \end{bmatrix} fname \begin{bmatrix} ; \\ SPACE \end{bmatrix} ext \ (x) \begin{bmatrix} \\ ALT \ MODE \end{bmatrix}$$

where,   a = A single letter, specifying a PIP operation.

        dd = the destination device

fname = file name

  ext = file name extension

  (x) = letter(s) specifying a PIP switch option(s).

  sd = source device

The left arrow (◄—) terminates information concerning the destination device. Data for the source device follows the ◄—. CR or ALT MODE must terminate a command string. ALT MODE forces PIP-9 to return control to KM-9 upon successful completion of the command. CR causes PIP-9 to wait for another command upon completion of the current one.

2

Multiple spaces are ignored by the command string processor. In fact, delimiters are absolutely required only following the operation character, device names and file names.

Example: T⎵DT1⎵NEWNAM BIN (B)◄─ ⎵DT2⎵OLDNAM⎵BIN⏎

OR T DT1:NEWNAM;BIN (B)◄─ ⎵DT2:OLDNAM;BIN⏎

The elements in the preceding example are:

T	PIP-9 Transfer File operation
DT1, DT2	DECtape 1 is the destination device, DECtape 2 is the source device.
NEWNAM, OLDNAM	File names
BIN	File name extension
◄─	Transfer direction indicator (right to left, i.e., DT2 to DT1)
B	Switch option

3.1 ⸱ Operation Character

The first character in a PIP-9 command string must be an operation character defining the main function to be performed. It must be followed by a space. Legal operational characters are listed in Table 3-1 below.

Table 3-1
PIP-9 Operation Characters

(T)	Transfer File	(V)	Verify File
(L)	List Directory	(S)	Segment File
(D)	Delete File	(B)	Block Copy
(C)	Copy	(N)	New Directory
(R)	Rename File		

3.2 Device Name

Because the PDP-9 ADVANCED Software System provides more than one device handler for some peripherals, a 2-letter mnemonic (corresponding to the first two letters of the handler name) is used for device name specification in PIP. Table 3-2 lists legal device names. For multi-unit peripherals, e.g., DECtape, the unit number, 0-7, appears after the device manemonic, e.g., DT7. The device name delimiter must be a colon (:) or a space.

Table 3-2
PIP-9 Device Names

(PR) Paper Tape Reader	(DT) DECtape
(PP) Paper Tape Punch	(MT) Magnetic Tape
(TT) Teletype	(CD) Card Reader
(LP) Line Printer	(DK) Disk

3.3   File Name and Extension

File name and extension, if used, constitute one element of the command string, where the file-name delimiter is a semicolon (;) or space. If the extension is omitted, the default assumption is three null characters. If more than one file name is specified, the second, third, etc., are separated from earlier names by commas (,). If the device is not a file-oriented device, file names may be omitted. Commas, however, must still be used for file count purposes. Some examples of device, file name and file-name extensions follow:

DT5:FILEA,FILEB;SRC	(2 files)	or DT5 FILEA, FILEB SRC
PR:,,	(3 files)	or PR ,,
PP:	(1 file)	or PP

A file name is a string of up to six (6) alphanumeric characters. Any printing character in the ASCII set may be used with the exception of a space, (:), (;), (,), ((and)), which have a specific delimiter meaning to PIP. The file-name extension may be up to three (3) characters long.

3.4   Switch Options

Switch options are enclosed in parentheses and require no delimiters to separate them from each other. They may appear either with the destination device information or with the source device information. PIP-9 switch options are divided into two classes: (1) data modes and (2) subsidiary operations.

3.4.1   Data Modes

(A)   IOPS ASCII
(B)   IOPS Binary
(I)   Image Alphanumeric
(H)   Image Binary
(D)   Dump

3.4.2   Subsidiary Operations

(G)   Correct Bad Parity/checksum lines
(E)   Convert tabs to spaces

4

(C)	Convert multiple spaces to tabs
(S)	Create new system directory
(N)	New directory
(Y)	Segment file
(W)*	Combine several source files, or tapes, stripping .EOT's and .END's from intermediate tapes.

or

(W)*	Comgine several binary files, stripping EOF's from intermediate files.

4. PIP-9 FUNCTIONAL DESCRIPTION

Functionally, PIP-9 may be described in terms of operations which may be specified and subsidiary switch functions requested as a part of a given operation. All PIP-9 operations and switches which are valid in the I/O Monitor paper tape system are also valid in the Keyboard Monitor system. The converse is not true, however.

4.1 Operations Under the I/O Monitor

Three PIP-9 operations are provided in an I/O Monitor environment: (1) Transfer File, (2) Verify File, (3) Segment File.

4.1.1 Transfer File (T) – T performs basic data or file transfer from one I/O device to another. In an I/O Monitor environment T is used to copy paper tapes and list paper tapes or card decks on the Teletype or line printer. T also provides the ability to create a source file by transferring from Teletype to paper tape punch. Paper tapes may be combined into one paper tape or segmented (IOPS ASCII tapes only) into several tapes.

4.1.2 Verify File (V) – The V operation allows parity and/or checksum verification of paper tapes. This function is particularly useful for verifying paper tapes copied with the T command.

4.1.3 Segment File (S) – The S operation provides a means for segmentation of source paper tapes whose unwieldy bulk makes two or more smaller tapes desirable. All PIP-9 operation commands are independent of other commands except Segment which is used prior to a Transfer command in order to specify at what points in the source file segmentation is to take place. The S command string allows for up to sixteen segmentation points or character strings (1-5 characters) at the beginning of lines at which segmentation is to take place. The file is terminated just prior to the segmentation point after a .EOT is appended. Transfer continues to the next segmentation point and so on.

---

* .END and .EOT on the final ASCII tape and EOF of the final binary tape are retained.

5

4.2   Switch Options Under the I/O Monitor

The data mode switches which may be used in an I/O Monitor environment are:

    (A)   IOPS ASCII

    (B)   IOPS Binary

    (I)   Image Alphanumeric

Function switches for use under the I/O Monitor are:

    (G)   Correct bad parity lines

    (E)   Convert tabs to spaces

    (C)   Convert multiple spaces to tabs

    (Y)   Segment file

    (W)   Combine files

Switch options may be used for some operations and are meaningless for other operations. Table 4-1 lists legal options by operation in an I/O Monitor environment. Furthermore, certain switch options conflict, e.g., combining the option to convert tabs to spaces (E) and spaces to tabs (C) is clearly a conflict. Table 4-2 lists legal switch combinations for the primary PIP-9 operation, Transfer File.

Table 4-1
Legal Operation/Switch Combinations

Operation	Legal Switches
Transfer File (T)	A,B,I,E,G,C,W,Y
Verify File (V)	A or B
Segment File (S)	(None)

Table 4-2
Legal Switch Combinations for Transfer File (T)

Switches	A	B	I	E	G	C	W	Y
E	✓			▨			✓	✓
G	✓			✓	▨	✓	✓	✓
C	✓				✓	▨	✓	✓
W	✓	✓		✓	✓		▨	
Y	✓			✓	✓			▨

4.2.1   Image Alphanumeric (I) - The (I) data mode permits copying of any paper tape but, in particular, (I) must be used when copying tapes which are in Hardware Read-in Mode (HRM or RIM tapes). Thus MACRO-9 .ABS or .FULL tapes require the (I) data mode.

4.2.2   IOPS Binary (B) - Relocatable binary tapes are reproduced using the binary data mode (B).

4.2.3   IOPS ASCII (A) - PDP-9 source tapes are normally copied using the (A) data mode. It should be noted, however, that use of the (A) mode will result in IOPS ASCII paper tapes having even parity in channel 8 of each frame. (See Section 2.1.2.1 of the PDP-9 Monitor Manual, DEC-9A-MAA0-D for a detailed discussion of IOPS ASCII.) If for some reason this is undesirable to the user, a data mode of (I) is recommended.

4.2.4   Bad Parity Correction (G) - Whenever data modes (A) or (B) are specified during a Transfer command, PIP-9 automatically verifies the correctness of parity and/or checksum. The G switch, used with IOPS ASCII mode only, allows the user to modify erroneous input lines via teletype keyboard input. User intervention may take one of three forms: (1) the line may be deleted, (2) the line may be accepted, or (3) the line may be replaced from the keyboard. The option to restart (↑P) is always available.

4.2.5   Tab to Space Conversion (E) - The E switch allows for conversion of horizontal tabs to spaces in order to allow off-line listing of ASCII tapes on Model 33 Teletypes. It is used with IOPS ASCII tapes. Since IOPS (Input/Output Programming System) follows a tenth position tab setting convention, enough spaces are substituted for a tab to place the next printing character of the line in position 10, 20, 30, etc.

4.2.6   Space to Tab Conversion (C) - In order to condense an ASCII paper tape the C switch is used to convert multiple spaces on an input file into horizontal tabs on the output file. Trailing spaces are simply deleted. Again, C is legal only when used with the (A) data mode.

4.2.7   Segment File (Y) - In order to apply the Segment operation during a Transfer file command, a (Y) switch is required in the T command string. On the basis of the (Y) switch the IOPS ASCII input file is segmented into the number of output files specified in the preceding S command.

4.2.8   Combine Files (W) - Although combining files or a series of paper tapes into one file is most common when Transferring from paper tape to a mass storage medium, it is possible to combine several small paper tapes into a single larger paper tape by indicating a W switch in a T command. Either IOPS

binary or ASCII tapes may be so combined. For binary files, all but the final EOF block of the input tapes are discarded on output. Likewise, when combining a series of IOPS ASCII paper tapes, all .EOT's and .END's are stripped except that of the final input tape.

4.3   Operations Under the Keyboard Monitor

The presence of mass storage devices in a PDP-9 configuration allows additional operations with PIP-9. In addition to the Transfer, Verify and Segment file operation, the following are available: (1) List Directory, (2) New Directory, (3) Delete File, (4) Rename File, (5) Copy Tape, and (6) Block Copy. (Additional switch options also become available.)

4.3.1   List Directory (L) - The directory of any file-structured mass storage device may be listed on teleprinter or line printer with the L command. The file name, extension, starting block number and number of blocks occupied, are printed along with the number of free blocks remaining.

4.3.2   New Directory (N) - The N command provides recording of a fresh directory on a mass storage device. In the case of DECtape, the File Bit Map blocks are cleared and the Directory block is initialized to indicate only the File Bit Map and Directory blocks as occupied.

4.3.3   Delete File (D) - To delete one or more named files from a mass storage device, the D operation is employed. Deletion implies removing references to the file from both the Directory and File Bit Map blocks.

4.3.4   Rename File (R) - Renaming one or more files requires an R command. Only the name and extension in the Directory are changed.

4.3.5   Copy Tape (C) - This function provides a convenient means of reproducing tapes (especially system tapes) in their entirety. Programmed read-after-write verification is performed. Differences in file structuring will be accounted for in the transfer from one type of device to another.

4.3.6   Block Copy (B) - The block copy operation is used with DECtape when copying one or more blocks seems desirable, e.g., when one or a few blocks on a tape seem suspect after a copy operation. The B operation obviates the need to recopy an entire tape. Blocks to be copied and verified are specified by their octal block number (0-1077).

4.4   Switch Options Under the Keyboard Monitor

Four additional switch options are available in a Keyboard Monitor environment. Two are data modes: Image Binary (H) and Dump Mode (D). Two are subsidiary functions: New Directory (N) and Create System Directory (S).

Tables 4-3 and 4-4 summarize legal switch/operation combinations within a Keyboard Monitor environment.

Table 4-3
Legal Operation/Switch Combinations

Operation	Legal Switches
Transfer File (T)	A,B,I,H,D,E,G,C,W,Y,N,S
Verify File (V)	A or B
Segment File (S)	(None)
List Directory (L)	N or S or None
New Directory (N)	(None)
Delete File (D)	(None)
Rename File (R)	(None)
Copy Tape (C)	N or S or H or None
Block Copy (B)	N or S or None

Table 4-4
Legal Switch Combinations for Transfer File

Switches	A	B	I	H	D	E	G	C	W	Y	N	S
E	✓					▨	✓		✓	✓	✓	✓
G	✓					✓	▨	✓	✓	✓	✓	✓
C	✓						✓	▨	✓	✓	✓	✓
W	✓	✓				✓	✓	✓	▨		✓	✓
Y	✓					✓	✓	✓		▨	✓	✓
N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	▨	
S	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		▨

4.4.1   Image Binary (H) - The reader is referred to Section 2.1.2 of the PDP-9 Monitor Manual for a discussion of data modes. The use of Image Binary as a data mode on mass storage devices such as DECtape or disk implies the intent to retain the exact form of the binary data as it originally appeared in hard copy, e.g., paper tape or cards such that, at a later time, the original data may be retrieved (once again onto paper tape), without alteration. It should be noted again that use of Image ASCII will always exactly reproduce an identical tape whether or not DECtape or disk has been used for intermediate storage (see Section 4.2.1).

The meaning of Image Binary as a switch option with the Copy (C) function is expanded beyond its customary meaning to imply a block by block DECtape copy. Later examples will illustrate this use of H mode.

4.4.2   Dump Mode (D) - Files recorded in dump mode may customarily be expected to reside on a mass storage device. Hence, D is used as a data mode most frequently when transferring to and from mass storage. There is no restriction on its use from mass storage to paper tape or vice versa, however.

4.4.3   New Directory (N) - The N switch option, like the N operation, initializes the Directory of the destination device. Permitting its use as a switch provides the added convenience of combining operations in a single command string.

4.4.4   Create System Directory (S) - The S switch constructs a basic system tape on the destination device prior to executing the main operation of the command string. The system tape must be mounted on unit 0 and DAT slot one (1) must be assigned to the system device in order to use the S switch. Basic system tape refers to the Directory, File Bit Maps, all absolute system programs and the relocatable system files: DDT9, .LOAD, .LIBR, INTEGE EAE, INTEGE NON, REAL EAE, and REAL NON.

5.   PIP-9 COMMAND STRING

This section illustrates PIP-9 commands and usage in detail. Since reference is made in earlier sections to I/O and Keyboard Monitor environmental differences, no further mention is made here. Examples are given without the optional (:) and semicolon (;) delimiters for use of which the reader may refer to Section 3, page 3.

5.1   Transfer File (T)

Under the T command are included the tasks of listing, copying, creating, combining and segmenting files. An input and an output device are required in the command string as well as one of the five (5) data modes. File names must be specified only for file structured devices.

5.1.1   Copying Files - The command:

T DT7 FILEA SRC (A)◄— PR

copies a single tape from the paper tape reader to DECtape unit 7 in IOPS ASCII mode.

The command:

T DT7 FA SRC,FB SRC,FC SRC (A)◄— PR,,

transfers three paper tapes as three separate files named FA SRC, FB SRC, and FC SRC.

The command:

T DT2 FILNEW BIN (B)◄——DT1 FILOLD BIN⟩

not only transfers FILOLD BIN from DECtape unit 1 to 2 but also renames the file: FILNEW BIN.

5.1.2   Creating Files - Creating a file is normally an Editor function. However, a T command from Teletype to any output device is perfectly legitimate. It should be kept in mind, however, that correction facilities provided by an Editor are not in PIP-9.

The command:

T PP (A)◄——TT⟩

directs PIP-9 to accept the input from Teletype to be punched on paper tape. To terminate file creation, a final line consisting of ↑D (Control Key D) must be typed.

5.1.3   Listing Files -

The command:

T LP◄——DK FILNAM SRC (A)⟩

lists FILNAM SRC on the line printer. IOPS ASCII is the only permissible mode to the line printer. Both IOPS and image ASCII are acceptable to the Teletype, the alternate listing device.

5.1.4   Using the G Switch - PIP-9 normally examines the correctness of parity and checksum when data mode A or B is specified. Transfer is discontinued after display of one of the two following messages on the Teletype:

                         INPUT PARITY ERROR
        or   INPUT CHECKSUM ERROR

The G switch allows for user correction of an ASCII line with bad parity. It may only be used with data mode A. Consider the following example:

T DT7 FILEA SRC (AG)◄——PR⟩

is typed. During transfer bad parity is encountered and the input parity error message is output on the Teletype followed by the line in error. The user may:

(1)   Accept the line by typing a carriage return.

(2)   Delete the line by typing D⟩

(3)   Retype the line, terminating with a carriage return.

(4)   Abort the operation by typing ↑P to restart PIP-9 or ↑C to reload the Keyboard Monitor.

5.1.5   Using the C or E Switches - The C or E switch may be used only with A as the data mode. C and E may not be used together.

The command:

T DT7 FILEA SRC (AC)◄——PR↓

effects a transfer from paper tape to DECtape during which process all multiple spaces are converted to tabs and trailing spaces are deleted.

The command:

T PP (AE)◄——DT2 FILEB SRC↓

effects a transfer of FILEB SRC from DECtape unit 2 to the paper tape punch during which process all tabs are converted to spaces allowing listing of the file on an off-line Teletype which lacks a tabbing mechanism.

5.1.6   Using the N or S Switch – Initializing the directory of certain mass storage devices, e.g., DECtape, is a frequent operation. The N switch allows initialization within the context of a File transfer. S is the only switch which conflicts with N.

The command:

T DT4 FILEA IMG (IN)◄——PR↓

initializes the Directory and File Bit maps of DECtape unit 4 and, subsequently, transfers the paper tape file to DECtape in image ASCII mode.

Given a DAT slot 1 assignment of DTA0 and PDP-9 Advanced Software System tape on DECtape unit 0, the command:

T DT4 FILEA BIN (BS)◄——PR↓

copies system programs and Directory information from DECtape unit 0 to unit 4 prior to transferring FILEA BIN from paper tape to DECtape.

5.1.7   Using the W Switch – Source files are frequently of such size as to require several paper tapes. Although they may be maintained on a mass storage device in segmented form, it is more often desirable to combine the segments into one file. The W switch performs this function. It is legal with data modes A or B and conflicts with the Y switch.

The command:

T DT1 FILEA SRC (AW)◄—— PR,,,, ↓

transfers five (5) ASCII paper tapes to DECtape unit 1 as the single file, FILEA SRC. Because intermediate .EOT or .END pseudo ops are no longer useful, all but the one on the final tape are deleted during transfer.

Used with a data mode of B, the W switch provides a convenient way to combine several binary subprograms into a single file such as a library file.

The command:

$$\text{T DT6 LIBRY BIN (BW)} \longleftarrow \text{DT1 A BIN,B BIN,C BIN}$$

combines the three (3) binary files, A BIN, B BIN and C BIN into one file LIBRY BIN, deleting intermediate End-of-Files in the process.

5.1.8   <u>Using the Y Switch</u> - In contrast to the W switch which combines files, the Y switch is used when ASCII file segmentation is required. It is used only with data mode A and conflicts with the W switch. Given a sizable source file on mass storage which is to be segmented, the command:

$$\text{T PP,,,,, (AY)} \longleftarrow \text{DT1 FILBIG SRC}$$

will result in FILBIG SRC being split up into six (6) paper tapes where five (5) segmentation points must have been specified in an S operation immediately preceding the current T command string (see Section 5.3, p.20).

The command:

$$\text{T DT3 FA SRC,FB SRC} \longleftarrow \text{DK FILBIG SRC (AY)}$$

similarly segments the disk file FILBIG SRC into two smaller DECtape files, FA SRC and FB SRC. The preceding S operation will have specified one segmentation point.

As each output file is closed, PIP-9 will output on Teletype ↑P which is the restart request. The purpose for this is to allow dismounting of tapes or removal of tape from the punch. Clearly, if the file was so large as to require segmentation, time for operator management of the segments seems appropriate.

5.2   <u>Verify File (V)</u>

File verification is performed in either IOPS ASCII (A) or IOPS binary (B) data modes. No other switch options apply to the verify operation. Since there is no output, only the input device (and file name if a file structured device) need be specified.

The command:

$$\text{V PR (B)}$$

requests parity and checksum verification of one binary paper tape. If a parity error occurs, the following message is typed:

<div align="center">INPUT PARITY ERROR</div>

If checksum failure:

<div align="center">INPUT CHECKSUM FAILURE</div>

For an ASCII file, the error line is also printed. In either case, after the message is printed, verification continues until the entire file has been examined allowing the user to assess how many errors are present.

Multiple files may be verified in a single command string. For example, the command

V DT3 FILEA SRC,FILEB SRC (A)↵

requests verification of both FILEA SRC and FILEB SRC.

5.3 Segment File (S)

The S operation allows specification of up to sixteen file segmentation points. Device names, file names and switch options are all meaningless in the S command.

The segmentation points are specified as 1 to 5 character strings. In the subsequent T command, if a Y switch is specified, PIP-9 will examine the beginning of every line for the specified segmentation points in order of occurrence. Vertical form control characters at the beginning of a line are excluded from the string search. As each segmentation point is found, PIP-9 will close the current output file segment, appending the pseudo op, .EOT, at the end of the segment. The next segment will start with the line which begins with the current segmentation point.

The command:

S TAGA,TAGB,TAGC,TAGD↵

sets up the four segmentation points TAGA,TAGB,TAGC, and TAGD for the immediately following command:

T PP,,,, (AY)◄—DT3 FILBIG SRC↵

the end result is five paper tapes, all but the last of which are terminated with .EOT. The last four begin with the lines TAGA---, TAGB---, etc.

As each segment is completed, ↑P will be output on the teleprinter allowing time to remove the paper tape segment, dismount tapes, etc. When ready to continue, the user simply types CTRL P and PIP-9 resumes segmentation.

5.4 List Directory (L)

The Directory contents of any file structured device may be listed by the L operation. The N or S switch options may be used.

The command:

L TT◄—DT1↵

results in a printout such as the one below:

```
           DIRECTORY LISTING
                               ──────── 1st Block of File
           MACRO ONE    4 ◢     226 ◄—# of Blocks Occupied
           MACRO TWO    5       140
           MACRO SRC    6       365
               4 SYS PGM BLKS
             121 FREE BLOCKS
```

5.5   New Directory (N)

Although the N function may be performed as a switch option in the command string of another operation, it has proven useful to include it as a distinct operation. No switch options are used with the N command and only the destination device need be specified.

The command:

N DT4⤸

results in a fresh directory on DECtape unit 4, a listing of which (as requested by an L operation) will appear as follows:

DIRECTORY LISTING

4 SYS PGM BLKS

1074 FREE BLOCKS

The four system program blocks are the Directory and three File Bit Map blocks.

5.6   Delete File (D)

File deletion is performed by the operation, D. Only the destination device is specified. No switch options are used.

The command:

D DT3 FILEA,FILEB⤸

causes PIP-9 to delete both FILEA and FILEB from DECtape unit 3.

5.7   Rename File (R)

The R command is used to rename files on a file-structured device without data transfer of any kind. No other unit is needed although the device name must appear with both source and destination data. A simple 9-character name substitution takes place into the directory entry section of the directory block. All switch options are illegal.

The command:

R DT2 NEWNAM BIN◄—DT2 OLDNAM BIN⤸

changes the name of the file OLDNAM BIN ON DECtape unit 2 to NEWNAM BIN.

5.8   Copy Tape (C)

Copying the contents of one file-structured device onto another implies one of two tasks: (1) incorporation of all information on the input device into the organization and content of the output device or (2) total replacement of all information on the output device by information on the input device. The latter is performed by the C operation in conjunction with the H data mode switch.

The command:

C DT5 (H)◄—DT3⤸

replaces all data on DECtape unit 5 with data from unit 3 in a block by block copy and read after write.

Incorporation is effected in one of the following three ways:

(1) Absence of switch options in the C command:

C DT5 ◄——DT3)

All files on DT3 will be incorporated into the file organization of DT5.

(2) Use of the N Switch:

C DT5 (N)◄——DT3)

Prior to transferring the files on DECtape unit 3 to unit 5 the Directory and File Bit Maps of unit 5 will be initialized.

(3) Use of the S Switch:

C DT5 (S)◄——DT3)

After copying the system from the system device onto DECtape unit 5, the files on unit 3 will be transferred to unit 5. PIP assumes that DAT slot 1 is assigned to the system device.

5.9   Block Copy (B)

To copy one or more blocks of one DECtape onto another, the B command is used. Switches N or S may be employed within command string also. Instead of specifying file names, actual octal block numbers (0-1077) are given in the command string. These block numbers may appear either with the destination or source data and are separated by commas.

The command:

B DT7◄—— DT4 5,15,165,1075)
or
B DT7 5,15,165,1075◄——DT4)

requests copy and verification of blocks 5,15,165 and 1075 from DECtape unit 4 to unit 7.

The command:

B DT2 10,15◄——DT1 4,3)

requests blocks 4 and 3 of DECtape unit 1 to be copied (and verified) onto blocks 10 and 15 of DECtape unit 2 respectively.

6.   CORRECTION PROCEDURES

Four correction procedures are available to the user in his operation of PIP-9. The procedure chosen is largely a function of what point in the PIP-9 process the user decides to correct or somehow modify PIP-9 action. Aborting a task is the most drastic procedure. Deleting one or more characters in a command string, negating the entire command string, or responding with corrected command string information after a PIP-9 error message are others.

6.1   ↑P (Control Key P)

A task may be aborted and PIP-9 restarted by typing in the ↑P (control key P) character at any time. ↑P has a secondary use in PIP-9 which is to indicate loading of the next in a series of paper tapes or output of the next in a series of files during segmentation. For example, at the end of each of several paper tapes to be combined into one output file, PIP-9 will type ↑P on the teleprinter directing the user to load the next paper tape. When ready, the user types ↑P for PIP-9 continuation.

6.2   Rubout (RO)

During typing of a command string, one or more characters may be deleted by use of the rubout (RO) key. For each character deleted, starting with the last one typed, a back slash (\) is echoed. For example:

T ZT3\\\DT3 FILEA (A)◄——PR⟩

The character Z is in error. Three rubouts have been used to back up to the erroneous character.

6.3   ↑U (Control Key U)

At any point while typing a command string, that is, prior to the CR or ALTMODE, a ↑U may be typed to delete the entire command string up to that point. An "at" sign (@) is echoed. The user may then start from the beginning of the command string again.

The command:

T ZT3@T DT3 FILEA (A)◄—— PR⟩

demonstrates a ↑U correction.

6.4   PIP-9 Error Detection and Correction

Once a command string is completed, PIP-9 may discover erroneous information. When this occurs, an appropriate error message is output to the teleprinter and the questionable command string is output up to but not including the offending character or element followed by "?", requiring correct completion by the user. If the user prefers to retype the command, a carriage return or ↑P will in this instance signal PIP-9 to accept a new command from the beginning. The characters RO and ↑U may not be used since the Teletype handler (which no longer has access to the erroneous command string) and not PIP-9, interprets and acts upon RO and ↑U.

Appendix II contains a complete list of PIP-9 error messages. Hence only two examples are cited here. Suppose a user intends to transfer an IOPS ASCII file from paper tape to DECtape. He types:

T DT2 FILEA SRC (F)◄——PR⟩

Recognizing F as an illegal switch option, PIP-9 types:

<div align="center">INVALID SWITCH OPTION</div>

<div align="center">T DT2 FILEA SRC(?</div>

The user may complete the command string from the erroneous character or element on to the end, or use a ↑P to indicate he prefers to restart the message.

If a DECtape handler and unit are not assigned to any of the positive .DAT slots, PIP-9 would type in response to the above example:

<div align="center">DEVICE (UNIT) NOT ASSIGNED TO POSITIVE .DAT TABLE</div>

<div align="center">T?</div>

to indicate that the command was in error (could not be honored due to absence of the necessary .DAT assignment) at the point of the device and unit specification code.

Appendix I
Summary of PIP-9 Commands

I/O Monitor Environment:

Command	Abbrev.	Dest. Dev.	Source Dev.	File Names	Legal Switches
Transfer File	T	Yes	Yes	No	A,B,I,E, G,C,W,Y
Verify File	V	No	Yes	No	A or B
Segment File	S	No	No	No*	None

Keyboard Monitor Environment:

Command	Abbrev.	Dest. Dev.	Source Dev.	File Names	Legal Switches
Transfer File	T	Yes	Yes	Yes	A,B,I,H,D,E, G,C,W,Y,N,S
Verify File	V	No	Yes	Yes	A or B
Segment File	S	No	No	No*	None
List Directory	L	Yes	Yes	No	N or S
New Directory	N	Yes	No	No	None
Delete File	D	No	Yes	Yes	None
Rename File	R	Yes	Yes	Yes	None
Copy Tape	C	Yes	Yes	No	N or S or H
Block Copy	B	Yes	Yes	No**	N or S

*Segmentation points instead of file names.

**Block numbers instead of file names.

Appendix II
PIP-9 Error Messages

Command String Too Long, Try Again	Retype command string
Ill. Function	Retype from function character on.
Ill. Dev. or Unit Ill. Dev. or Unit Terminator Dev. Ill. for Option or Function and Direction	Retype from device name on.
Dev. (Unit) Not in + DAT Table Ill. Sys. Dev. in DAT Slot 1	Type ↑C to restore Monitor and perform ASSIGN
Sys. Tape Not on Unit 0	Mount System Tape on Unit 0 and retype command string.
Too Many Files or Blks., Try Again	Retype command string
Too Many Chars. in File or Ext. Name Source File Not on Dev.	Retype from File Name on.
Too Many Source Files Too Many Dest. Files	Check number of files actually transferred and type another command string to transfer remainder.
Data Mode Needed	Type data mode in parentheses followed by carriage return
Switch Ill. for Dev. Ill. Switch Switch Conflict Switch Ill. for Function	Retype from switch on.
Ill. Terminator	Retype from terminator on.
Input Parity Err.	If binary, check data. If ASCII, retype command string using G switch.
Input Checksum Err ASCII Input Line Too Long	Check data
Ill. Blk.#	Retype from block # on.
Read – Comp. Err. on Blk. n	When operation complete, try B function on error block.
S Operation Not Performed	Execute S operation; then retype T command.
Strings 1 to 16 Accepted	Perform segmentation; then further segment last destination file.
Too Few Dest. Files for # of Segment Points	Retype command string with correct # of destination files. (1 more than # of segmentation points).

LINKING LOADER

LINKING LOADER

CONTENTS

SECTION 1
INTRODUCTION

This document describes the operation of the Linking Loader and the composition of the binary information which comprises a loadable program unit. Operating procedures for the I/O Monitor and Keyboard Monitor environments are included along with memory maps of the various phases of loading by the Linking Loader.

SECTION 2
DESCRIPTION

The Linking Loader loads and links relocatable or absolute binary program units as produced by the FORTRAN IV compiler and the MACRO-9 Assembler. Absolute and relocatable coding should not be intermixed in one unit, and care should be taken in linking relocatable and absolute units. For FORTRAN and Assembler generated program units, the Loader also assigns the common data storage area. The input medium may be any input device.

Initially the loader will load all the program units whose names appear on the command string (see operating procedures, section 7). After all the programs named by the command string have been loaded, the Loader automatically loads and links all requested and unresolved library subprograms. The requested library subprograms are loaded from the external library and the system library (in that order). After both libraries have been examined for requested subprograms, the loader displays the names of all subprograms which have not been found. If the user requires I/O handlers that are already in core for Linking Loader purposes, the resident handlers will be used.

As individual program units cannot be executed if the program flows across an 8K memory bank, the Loader will prevent this type of loading. The Loader will, however, load (and link) the program in the next memory bank. No checking of this type is made with absolute binary program units.

Optionally, symbols and their absolute definitions are loaded into a program dictionary for use by the on-line debugging package (DDT). The loader also sets up for DDT the start execution address of the main program (in the system communication tables) and the initial relocation value of all the program units.

SECTION 3
INFORMATION UNITS

The binary output from the FORTRAN compiler and the MACRO-9 Assembler consists of blocks of information units. Each information unit consists of an identification code (6 bits) and a data word (18 bits). The form of the object program at run time is determined by the content and the ordering of the information units. Several information units may be grouped to convey a single run-time instruction to the Loader.

A block of information units consists of four 18-bit machine words arranged in the following manner:

	0 5	6 11	12 17
Word 1	Code 1	Code 2	Code 3
Word 2	Data Word 1		
Word 3	Data Word 2		
Word 4	Data Word 3		

Standard IOPS binary line sizes (48 information words and a 2 word header) are input by the Loader.

SECTION 4
IDENTIFICATION CODES

The identification code is used to instruct the Loader on how to handle the associated data word.

Code	Loader Action
01	Program Unit Size

The data word specifies the number of machine words required by this program unit. This number does not include the required number of machine words for common storage. The program size is used by the Loader to determine whether the program will fit within the unused locations of any available 8K memory bank. Loading terminates with an appropriate error message if the program cannot be loaded. This information unit appears only once per program unit and is the first information unit of the binary output. In absolute loads, no checking is made for overlays; this is left to the discretion of the user. The program size is also used to determine where to begin loading as loading proceeds from the top of core down (see Memory Maps).

```
        0  1  3                        17
       +--+--+----------------------------+
Data Word | X |  |      Program Size        |
       +--+--+----------------------------+
        ↑  { 1 if absolute load
           { 0 if relocatable load
```
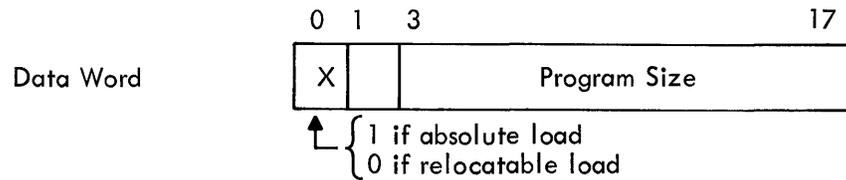
02	Program Load Address

The data word is an unrelocated memory address. This address specifies either an absolute or a relative storage address for program data words and is incremented by one for each data word stored (codes 03, 04, and 05). If the address is relative, it is initially incremented by the current relocation factor (modulo 15 bits). Bit 0 of the data word is used to indicate an absolute address (bit 0 = 1) or a relative address (bit 0 = 0).

```
        0  1 2 3                       17
       +--+---+---------------------------+
Data Word | X | 0 0 |     Load Address       |
       +--+---+---------------------------+
        ↑      { 0, relative load address
               { 1, absolute load address
```
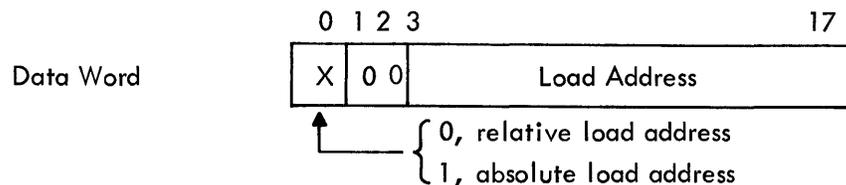
03	Relocatable Instruction

The data word is a memory referencing instruction. The address portion of the instruction is incremented by the current relocation factor (modulo 13 bits).

Code	Loader Action

The instruction is stored in the location specified by the load address which is incremented by one after the word is stored.

```
        0         4 5                    17
        ┌───────────┬──────────────────────┐
Data Word │  Op Code  │     Unrelocated       │
        │           │   Memory  Address     │
        └───────────┴──────────────────────┘
```
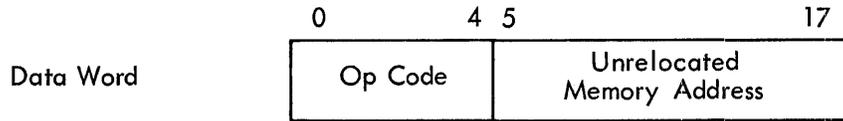
04   Absolute Instruction/Constant/Address

The data word is either a non-memory referencing instruction, a non-reloca-table memory referencing instruction, an absolute address, or a constant. The word is stored in the location specified by the load address which is incre-mented by one after the word is stored.

```
        0                                17
        ┌──────────────────────────────────┐
Data Word │        Non-Relocatable Word       │
        └──────────────────────────────────┘
```
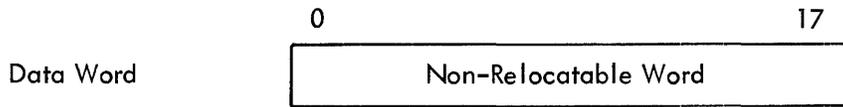
05   Relocatable Vector

The data word contains a relocatable program address (vector). The word is incremented by the current relocation factor (modulo 15 bits). The data word is stored in the location specified by the load address which is incremented by one after the word is stored.

```
        0         2 3                    17
        ┌─────────────┬────────────────────┐
Data Word │ 0   0   0 │       Vector         │
        └─────────────┴────────────────────┘
```
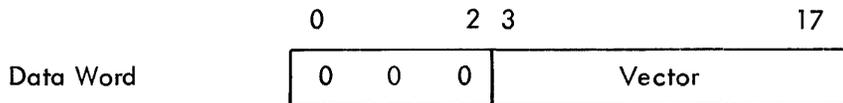
06   Non-Common Storage Allocation

The data word specifies the number of machine words required for non-common variable and array storage. Storage allocation begins at the address specified by the load address. The load address is incremented by this number. The block of memory is not cleared.

```
        0         4 5                    17
        ┌───────────┬──────────────────────┐
Data Word │ 0 ──── 0 │     Storage Size      │
        └───────────┴──────────────────────┘
```
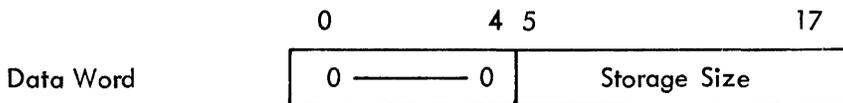
07   Symbol-First Three Characters

The data word contains the first 3-characters of a symbol in radix 50_8 format (see appendix 1). The data word is saved by the loader for future reference.

Code	Loader Action

```
            0 1 2                        17
           ┌──┬──┬─────────────────────────┐
Data Word  │X │0 │        Symbol           │
           └──┴──┴─────────────────────────┘
            ↑  ┌ 0, 1- to 3-character symbol
            └─ {
               └ 1, 4- to 6-character symbol
```
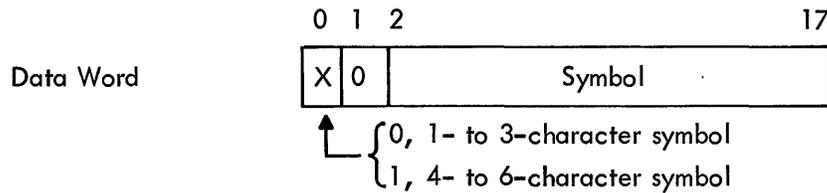
08   Symbol – Last Three Characters

The data word contains the last 3-characters of a symbol in radix 50_8 format. The data word is saved by the loader for future reference. This word is used only if in the code 07 data word bit 0 = 1.
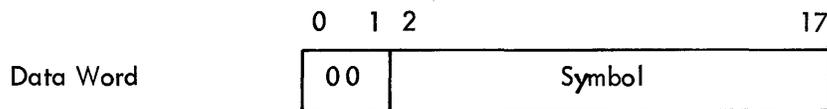
```
           0    1 2                        17
           ┌─────┬─────────────────────────┐
Data Word  │0  0 │        Symbol           │
           └─────┴─────────────────────────┘
```

09   External Symbol Definition

The data word contains the unrelocated address of the transfer vector for the subprogram named by the last symbol loaded (codes 07 and 08). If the external subprogram has already been loaded, the address (definition) of the symbol is stored into the specified vector address (relocated modulo 15 bits). If the subprogram has not been loaded and this is the initial request, the symbol and the relocated (modulo 15 bits) transfer vector address are entered into the Loader symbol dictionary as a request for subprogram loading. This action automatically forces the Loader into a library search mode when the end of the command string is encountered. If the Loader is already in the library search mode, it remains there until all virtual globals have been resolved. If the subprogram has been previously requested (symbol in dictionary) but not loaded, the Loader chains the reference locations. This chain, generated exclusively by the Loader, is followed when the external definition is encountered. (Unchained transfer vector locations must initially contain a reference address (code 04 or 05) to themselves.) For example, ·GLOBL SUB where SUB is virtual should cause the output of

```
          0 1 2                        17
         ┌──┬──┬──────────────────────────┐
07       │0 │0 │      SUB(radix 50₈)      │
         ├──┴──┴──────────────────────────┤
09       │           TVADD                │
         └────────────────────────────────┘
           5                            17
                         :
          0    2 3                      17
         ┌─────┬──────────────────────────┐
05       │     │          TVADD           │   TVADD
         └─────┴──────────────────────────┘
```
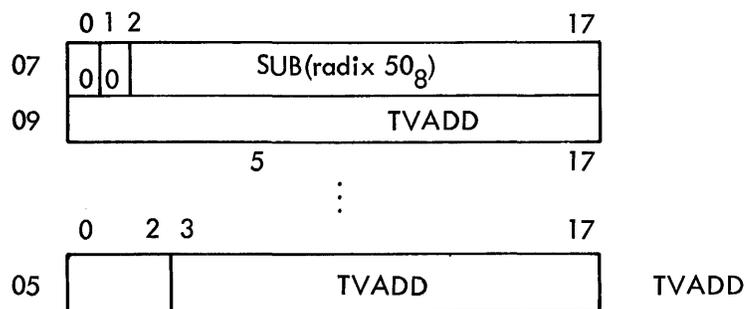
and SUB defined internally as TVADD. Subroutine calls are made via

JMS* SUB

Code | Loader Action

	0	3		17
Data Word	0		Transfer Vector Address	
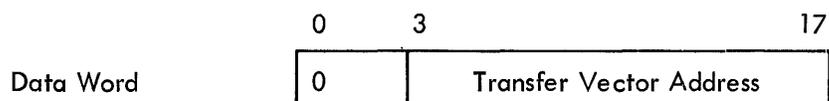
10   Internal Global Symbol Definition

The data word contains the unrelocated or absolute address (definition) of the last symbol loaded (codes 07 and 08). The last symbol loaded is a global symbol internal to the program unit which follows. In the library search mode, if a request for subprogram loading exists (code 09) in the Loader dictionary, the relocated (modulo 15 bits) definition is stored in the specified transfer vectors and the program unit is loaded. The definition also replaces the transfer vector address in the Loader dictionary. If no request for loading exists, the program unit is not loaded and the Loader continues to examine information units until the next internal global symbol definition is found (library search mode). If the program unit is to be loaded, all internal symbols following the one causing loading are automatically entered into the Loader dictionary as defined global symbols. If the symbol already exists in the dictionary and is defined (indicating that a program unit with the same name is already loaded) the current program unit is ignored.

	0	3		17
Data Word	0		Symbol Definition	
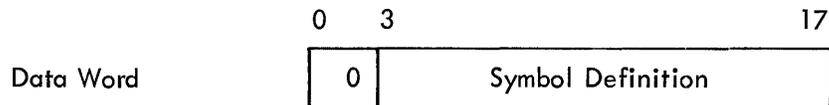
11   Block Data Declaration

This information unit instructs the Loader that the common blocks and data constants following are part of a block data subprogram.

	0	3		17
Data Word	0		Block Size	
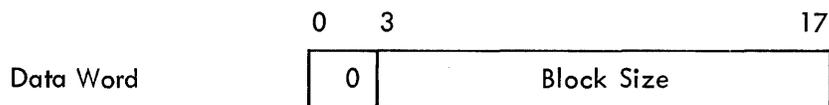
12   Common Block Definition

The data word specifies the number of machine words required for the common block named by the last symbol loaded (codes 07 and 08). In general, the assignment of memory space for the common block is deferred until all requested library and subprograms have been loaded. The exception to this rule occurs when the block data declaration (code 11) has been encountered. In this case the common block name is treated as an internal global symbol and the block is assigned to memory. After the block is assigned to memory, the starting address is entered into the Loader dictionary and the starting address is saved by the Loader for future use (code 13). All symbols in the dictionary associated

Code	Loader Action

with the block are assigned addresses with respect to this starting address. All symbols which are yet to be loaded (via code 13 and 14) will also be assigned as they are encountered. When the block data flag is not set, the Loader enters the name and the size into the dictionary (if it is not already there) and also enters the word containing the next available dictionary entry address. This entry will contain the first symbol in this common block and will be used as the head of the chain of all symbols in this common block. The address of the head of chain is saved by the Loader so that the new set of symbols in the common block may be added to the chain. The larger of the two block sizes is retained as the block size.

When the common block has been assigned memory locations, the assigned address is saved by the Loader for future reference (code 13) and the respective lengths are compared. Loading terminates, with an appropriate error message, if the assigned block is the smaller. When the assigned block is larger or both are equal, loading continues.

```
        0   3                    17
Data Word  | 0 |    Block Size      |
```
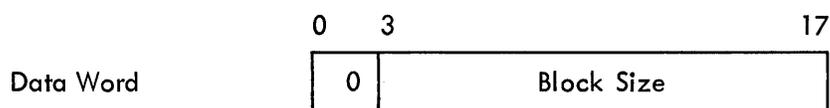
13 Common Symbol Definition

The data word specifies the relative location of the last symbol loaded (codes 07 and 08) in the last common block (code 12). If the associated common block has been defined (block data), the absolute address of the symbol is calculated (block address plus relative position) and placed in T. V. location (code 14). When the common block has not been assigned, the relative address is entered into the Loader dictionary. and chained to the symbols associated with the common block.

```
        0   3                    17
Data Word  | 0 |  Relative Address   |
```
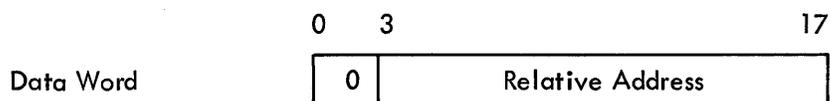
14 Common Symbol Reference Definition

The data word contains the unrelocated address of the transfer vector for references to the common symbol named by the last symbol loaded (codes 07 and 08). The symbol definition (code 13) is stored in the relocated (modulo 15 bits) address specified when the associated common block has been assigned (code 12). When the block has not been assigned, the relocated (modulo 15 bits) address is entered into the Loader dictionary along with the relative address (code 13) of the symbol.

Code	Loader Action

```
         0   3                          17
        ┌───┬──────────────────────────┐
Data Word│ 0 │     Address of Vector    │
        └───┴──────────────────────────┘
```
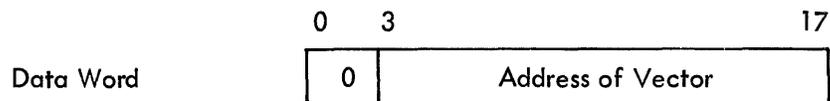
15   **Data Initialization Constant – First Word**

The data word contains the first machine word of a data initialization constant. It is saved by the Loader for future use (code 18).

```
         0                             17
        ┌──────────────────────────────┐
Data Word│        Data Constant         │
        └──────────────────────────────┘
```
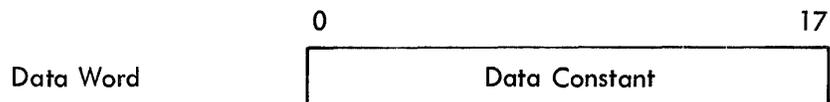
16   **Data Initialization Constant – Second Word**

The data word contains the second machine word of a data initialization constant. It is saved by the Loader for future use (code 18).

```
         0                             17
        ┌──────────────────────────────┐
Data Word│        Data Constant         │
        └──────────────────────────────┘
```

17   **Data Initialization Constant – Third Word**

The data word contains the third machine word of a data initialization constant. It is saved by the Loader for future use (code 18).

```
         0                             17
        ┌──────────────────────────────┐
Data Word│        Data Constant         │
        └──────────────────────────────┘
```
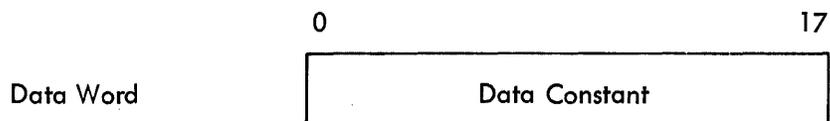
18   **Data Initialization Constant Definition**

The data word contains the <u>relative</u> load address of the last data initialization constant loaded (codes 15, 16, and 17) and a mode code identifying the constant (real, integer, double, logical). The load address is incremented by the current relocation factor (modulo 15 bits) if the constant initializes a non-common storage element. When the constant initializes a common storage element (indicated by the presence of the block data flag, code 11), the load address is incremented by the address of the last common block loaded (code 12). The constant is stored according to mode and the relocated load address.

```
              0 1 2 3                    17
             ┌─┬──┬───────────────────────┐
Data Word    │0│XX│     Load Address        │
             └─┴──┴───────────────────────┘
                ↑  ┌ 00, mode = integer (1 word)
                └──┤ 01, mode = real     (2 words)
                   │ 10, mode = double  (3 words)
                   └ 11, mode = logical (1 word)
```
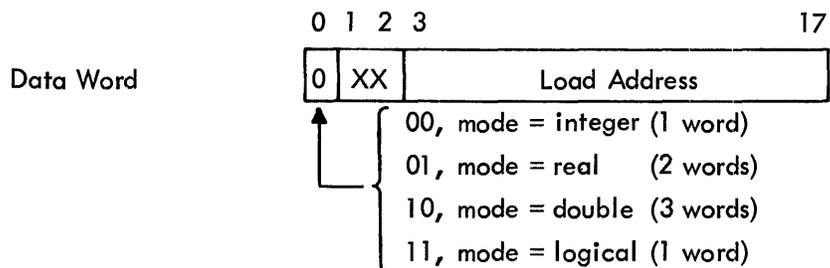
Code	Loader Action

19 Internal Symbol Definition

The data word contains the unrelocated or absolute address (definition) of the last symbol loaded (codes 07 and 08). The symbol is strictly internal to the program being loaded and is entered conditionally (if a DDT Load) along with its relocated address (modulo 15 bits), into the DDT symbol dictionary. The program unit name is indicated by bit 0 of the data word.

```
         0 1   3                        17
        ┌──┬──┬──────────────────────────┐
Data Word│X │0 │    Symbol Definition     │
        └──┴──┴──────────────────────────┘
         ↑   ┌ 0, internal symbol
         └──{
             └ 1, program name -- from FORTRAN IV or
                                  MACRO-9 command string
```
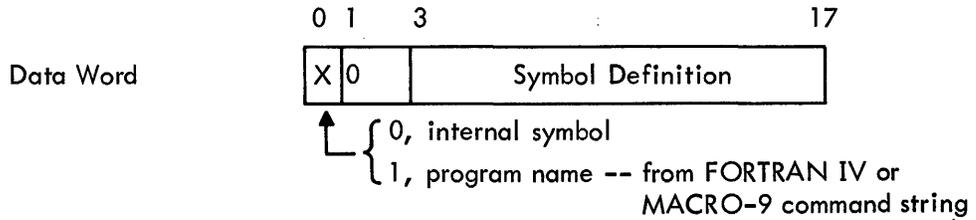
All symbols fall into this category.

20 String Code - First Half

The data word contains the unrelocated address of a data word whose address portion is to be replaced by another value. The relocated (modulo 15 bits) address is saved by the Loader for future use (code 21).

```
         0     3                  17
        ┌──────┬──────────────────┐
Data Word│  0   │  String Address  │
        └──────┴──────────────────┘
```
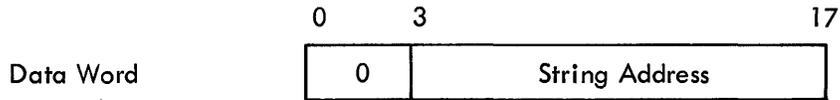
21 String Code - Second Half

The data word contains an unrelocated address. The address portion of the data word specified by the first half-string code (code 20) is replaced with this address (relocated modulo 13 bits).

```
         0       4 5              17
        ┌────────┬──────────────────┐
Data Word│ 0────0 │Replacement Address│
        └────────┴──────────────────┘
```
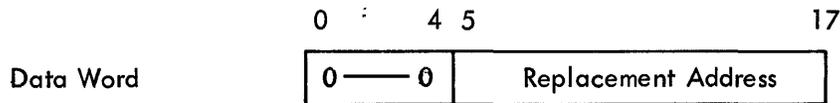
22 Input/Output Device Routine Request

The data word specifies the unit number (. DAT slot number) associated with a device level I/O routine. The Loader defers loading of any I/O routines until all other subprogram loading has been completed; when subprogram loading is complete , the system library is searched for all requested I/O device routines not already residing in memory (see Operating Procedures). The I/O routines are then loaded.

Code	Loader Action

```
            0              9              17
Data Word   ┌──┬──────────┬──────────────┐
            │  │          │ Unit Number  │
            └──┴──────────┴──────────────┘
```
↑ { 0 = single unit   ↑— 2's complement when
    1 = all units   negative
    (·IODEV ALL) all
    positive ·DAT slots with non-zero contents

23   End of Program Unit

This information unit is the last unit of a program unit. The data word contains
the unrelocated start execution address of the program. This address is relo-
cated (modulo 15 bits) and entered into the system communication tables to be
used when control is given to the user. Only the first start address encountered
is entered into the communication tables. (It is assumed that the first program
unit specified in the command string is the main program.) The first address of
the main program will be used if the . END pseudo-op did not have a start ad-
dress. When loading from either the system or external libraries, the end unit
causes the Loader to examine the next line buffer for the end-of-file condition.
When the end-of-file for the external library is obtained, the Loader automati-
cally begins searching the system library to resolve any remaining globals.
Upon encountering the end-of-file of the system library, the Loader announces
any unresolved global names. When loading is complete, control is returned
to the Monitor for dispatching to the user, DDT, keyboard listener (see Opera-
ting Procedures).

```
            0   3                      17
Data Word   ┌───┬──────────────────────┐
            │   │    Start Address      │
            └───┴──────────────────────┘
```

SECTION 5

MAIN PROGRAM ORGANIZATION

PROGRAM SIZE (code 01) absolute or relative, does not include COMMON size

PROGRAM NAME (code 19)

PROGRAM LOAD ADDRESS (code 02) absolute or relative

COMMON STORAGE (codes 12, 13 and 14)

NON-COMMON STORAGE (code 06)

    Array Declaration Information

    Equivalenced Arrays and Variables

    Non-Equivalenced Arrays

PROGRAM BODY

	Codes	Codes	
Instructions	03	07 }	Symbol
	04	08 }	
	05	09	External Symbol Definition

    Constants

    Non-Equivalenced Variables

    Literals

    Transfer Vectors (code 05)

EXTERNAL SYMBOL DEFINITIONS (code 09)

END (code 23)

5.1 SUBPROGRAM ORGANIZATION

PROGRAM SIZE (code 01) absolute or relative, does not include COMMON

INTERNAL GLOBAL DEFINITIONS (code 10)

PROGRAM NAME (code 19)

PROGRAM LOAD ADDRESS (code 02) absolute or relative

COMMON STORAGE (codes 12, 13 and 14)

NON-COMMON STORAGE (code 06)

    Array Declaration Information

    Equivalenced Arrays and Variables

    Non-Equivalenced Arrays

PROGRAM BODY

	Codes	Codes
	03	07
Instructions	04	08
	05	09

    Constants

    Non-Equivalenced Variables

    Literals

    Transfer Vectors (code 05)

EXTERNAL SYMBOL DEFINITIONS (code 09)

END (code 23)

5. 2   BLOCK DATA SUBPROGRAM ORGANIZATION

        BLOCK DATA INDICATOR (code 11)
        PROGRAM NAME (code 19)
        COMMON STORAGE (codes 12, 13, and 14)
        DATA INITIALIZATION CONSTANTS (codes 15, 16, 17, and 18)
        END (code 23)

SECTION 6
DEFINITIONS

Loadable Program Unit	A main program, subprogram, or a block data subprogram.
Transfer Vector	A core location containing the address of a subprogram or an entity in common. All references to subprograms and entities in common are indirect.
Internal Global Symbol	A symbol whose definition is accessible to all programs.
External Symbol	A symbol which is referenced in one program and defined in another.
Relocation Factor	The amount added to relative addresses to form absolute addresses; initially, the first loadable core location. The relocation factor for programs following the first program unit is the next available load address.
Radix 50_8 Format	A method of symbol concatenation utilizing 50_8 characters as a "number" set each with a unique value between and including 0 to 47_8. The symbol ("number") is converted using standard base conversion methods (see appendix 1).

SECTION 7
LINKING LOADER OPERATING PROCEDURES

7.1   I/O MONITOR ENVIRONMENT

When the Linking Loader is ready to accept the load command string from the keyboard, it will output to the teleprinter.

LOADER

> Set up the input device and if it is the paper tape reader, momentarily depress the tape feed control to clear the reader out-of-tape flag.

The file names, of all the programs that are to be unconditionally loaded from the input device (· DAT Slot · -4) must be input from the Teletype Keyboard, in the following form:

>NAME1, NAME2, NAME3 ↓

>NAME4, NAME5 (ALT MODE)

The main program must be requested first. The file names consist of 1 to 6 characters with any characters over 6 being ignored. File names are exactly those used in command strings for assembly or compilation.

A file name is terminated by a comma (,), a carriage return (↓), or ALT mode. Until the comma, the carriage return, or the ALT mode is encountered, N RUBOUTS may be used to delete the N previous characters of the file name.

ALT mode terminates the command string. When the input device is not file oriented, N commas, followed by the ALT mode will prime the Loader to load N + 1 programs from the device.

After loading the programs requested in the keyboard command string, the Loader will attempt to resolve all unsatisfied subroutine requests by scanning the system library (· DAT Slot -1).

The library must be in the following format:

7.1.1   Structure of System Library

ONE FILE: .LIBR

| PROGRAM UNIT | PROGRAM UNIT | PROGRAM UNIT | — — — — — | PROGRAM UNIT | END-OF-FILE UNIT |

PROGRAM UNIT:

| PROGRAM SIZE DESCRIPTOR |
| INTERNAL GLOBALS |
| LOAD ADDRESS DESCRIPTOR |
| DATA |
| VIRTUAL GLOBALS |
| END CODE |

} PROGRAM UNIT

$C_n = 23_{10}$ TERMINATES A PROGRAM UNIT. THE NEXT UNIT MUST BEGIN A NEW IOPS BINARY BUFFER.

PROGRAM UNIT:

| ONE IOPS BINARY BUFFER | — — — — — | ONE IOPS BINARY BUFFER |

| H_1 | } TWO-WORD HEADER |
| H_2 |
| $C_1 / C_2 / C_3$ | } FOUR-WORD GROUP |
| D_1 |
| D_2 |
| D_3 |
| $C_4 / C_5 / C_6$ | — DESCRIPTOR |
| D_5 |
| D_6 |
| I |
| I |
| I |
| $C_{34} / C_{35} / C_{36}$ |
| D_{34} |
| D_{35} |
| D_{36} |

48_{10} WORDS

0-89-17

| WORD 0 | 1 | 101 | } END-OF-FILE UNIT |
| WORD 1 | CHECK – SUM |

END-OF-FILE UNIT ONLY PRESENT AT END OF .LIBR FILE. MUST BE REMOVED FROM END OF ALL PROGRAM UNITS, SINCE FORTRAN IV AND MACRO-9 ALWAYS CREATES E-O-F UNIT.

7.1.2   Loader Memory Map

The loader will output to the teleprinter the names and relocation factors (starting load addresses) of all the programs requested in the command string, followed by the required library routines in the following format:

```
NAME1        16572
NAME2        14301
NAME3        10765
NAME4        06427
NAME5        06313
LIBR1        05304
LIBR2        04112
```

NOTE: Whenever the Loader detects end-of-medium in the input device, or the system library device, it types ↑P on the teleprinter. To continue, place more input in the device, and if the paper tape reader momentarily depress the tape feed control, and type ↑ P on the keyboard.

7.1.3   Error Messages

The Loader will output to the teleprinter ·LOAD followed by the pertinent error code and then it will halt.

Error Code	Meaning
1	Memory overflow – the Loader's symbol table and the user's program have overlapped. The loader memory map will contain printouts of all programs successfully loaded, prior to the one which caused the memory overflow. Use of COMMON storage may enable the program to be loaded as it can overlay the Loader and its symbol table because it is not loaded into until run time.
2	Input Data Error – parity error, checksum error, illegal data code or buffer overflow (input line bigger than Loader's buffer).
3	Unresolved globals – if an explicitly or implicitly requested program cannot be found, it will appear in the memory map with an address of 00000. This indicates that loading was unsuccessful; the cause of the trouble should be remedied and loading tried again.
4	Illegal ·DAT slot request–the ·DAT slot requested is (a) out of the range of legal ·DAT slots (b) 0 (c) does not have a device associated with it; that is, it was not set up at SYSTEM generation time, and (in Keyboard Monitor systems) was not set up by a ASSIGN command.

When all the requested programs have been loaded and all library requests satisfied, the Loader will output ↑S on the teleprinter and sit in a JMP loop. Typing ↑S on the keyboard will give control to the starting address of the user's main program.

NOTE: If use is to be made of the paper tape reader, load the reader and then momentarily depress the tape-feed control.

When the user program has completed its operation and terminated via the ·EXIT command, the computer will halt.

If a DDT load, on completion of the loading and building of a DDT symbol table (exclusive of the library routine symbols and those of DDT itself) control is automatically given to the starting address of DDT. DDT types DDT to inform the user that it is waiting for a DDT command.
>

The user can force control back to DDT whenever he wants, by typing ↑T on the Teletype keyboard.

7.2   KEYBOARD MONITOR ENVIRONMENT

The operating procedures noted below are required in addition to those described under the I/O Monitor environment.

After loading the programs requested in the keyboard command string, the Loader attempts to resolve all unsatisfied subroutine requests by scanning the external (· DAT Slot -5) and system (· DAT Slot -1) libraries, in that order.

In order to inform the Loader that an external (user) library file exists for this load, it is necessary to ASSIGN an I/O device to · DAT Slot -5 prior to the LOAD, DDT, DDTNS or GLOAD command, i.e.,

```
$ASSIGN    DTA4    -5
$LOAD
```

The format of the external library file is identical to that of the system library file (see section 7.1.1).

If a DDT load, (DDT), on completion of the loading and the building of a DDT symbol table (exclusive of the library routines' symbols and those of DDT itself), control is automatically given to the starting address of DDT.

DDT uses · DAT slots -6 and -10 for patch output and patch input respectively. If the user knows that he will not make use of this feature, he should ASSIGN NONE to those slots so that un-necessary device handlers do not take up needed core space. For example,

```
$ASSIGN    NONE    -6, -10
$DDT
```

A program may be loaded with DDT but without the DDT symbol table by requesting loading with the DDTNS keyboard command. For example,

```
$ASSIGN    NONE    -6, -10
$DDTNS
```

This feature gives the user more operating space but deprives him of symbolic references to user symbols in DDT commands.

If a loading error occurs, an appropriate error message will be output to the teleprinter and control will be given to the system bootstrap to reinitialize the Keyboard Monitor.

When all the requested programs have been loaded and all library requests satisfied, the Loader will

    a.   If LOAD, wait on the recognition of ↑S by the keyboard handler and then give control to the starting address of the user's main program.

    b.   If GLOAD, give control to the starting address of the user's main program.

    c.   If DDT or DDTNS, automatically give control to the starting address of DDT.

When the user program has completed its operation and terminated via the · EXIT command, control will be given to the system bootstrap to reinitialize the Keyboard Monitor and wait for the next keyboard command.

SECTION 8
MEMORY MAPS

8.1   I/O MONITOR ENVIRONMENT

Linking Loader Tape

8K or 16K or
24K or 32K

BOOTSTRAP LOADER IN HRM FORMAT

● SCOM AND ● SCOM +3

USER PROGRAMS
↓
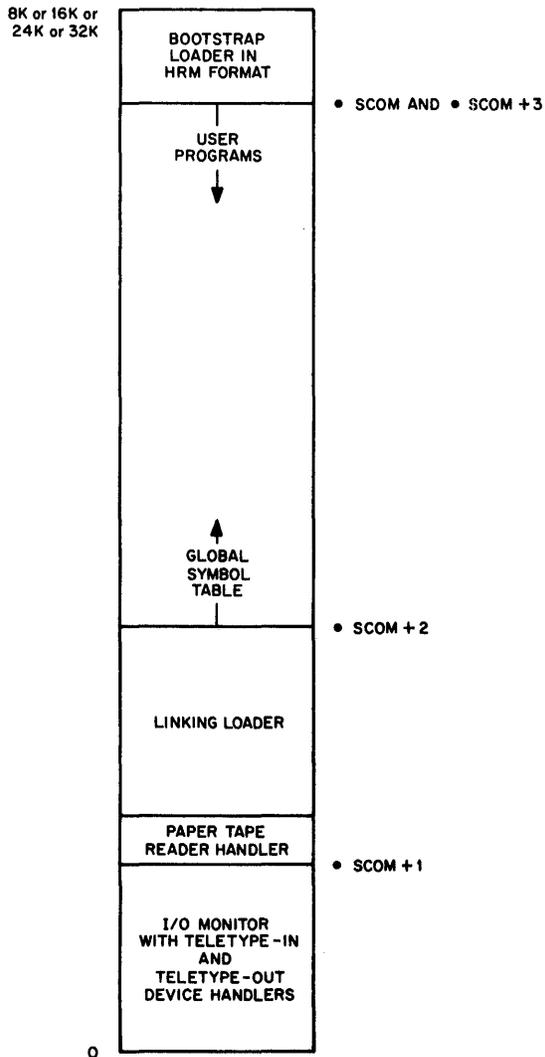
Refer to memory map 2A of Keyboard Monitor Systems for results of Link Loading.

↑
GLOBAL SYMBOL TABLE

● SCOM +2

LINKING LOADER

PAPER TAPE READER HANDLER

● SCOM +1

I/O MONITOR WITH TELETYPE-IN AND TELETYPE-OUT DEVICE HANDLERS

0

DDT Tape

8K or 16K or
24K or 32K

BOOTSTRAP LOADER IN HRM FORMAT	
	● SCOM
DDT	
	● SCOM +3
USER PROGRAMS ↓	
↑ GLOBAL AND DDT SYMBOL TABLES	
	● SCOM + 2
LINKING LOADER	
PAPER TAPE PUNCH HANDLER	
PAPER TAPE READER HANDLER	
	● SCOM + 1
I/O MONITOR WITH TELETYPE-IN AND TELETYPE-OUT DEVICE HANDLERS	

0
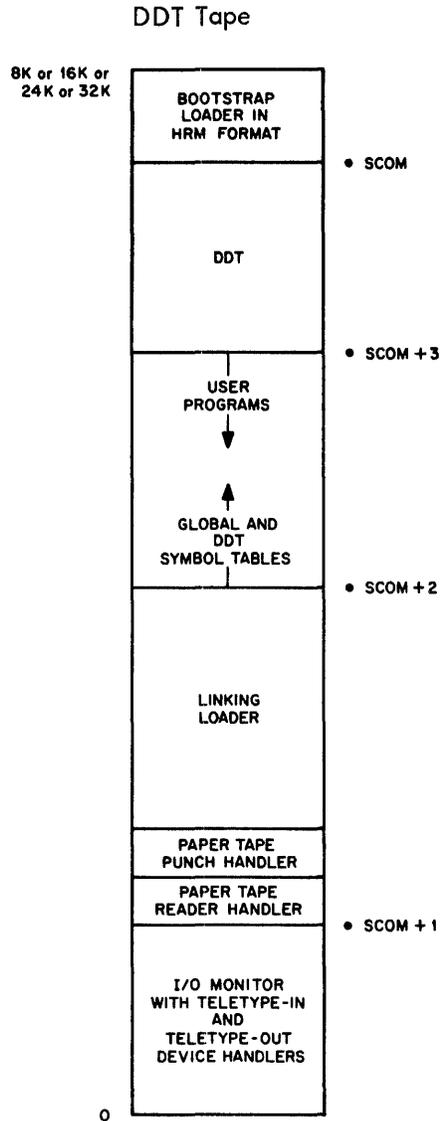
Refer to memory map 2B of Keyboard Monitor Systems for results of Link Loading in DDT mode.

Paper Tape Punch Handler is only present in version of DDT with patch file capabilities.

8.2   KEYBOARD MONITOR ENVIRONMENT

LOAD
GLOAD
DDT
DDTNS (DDT without symbol table)
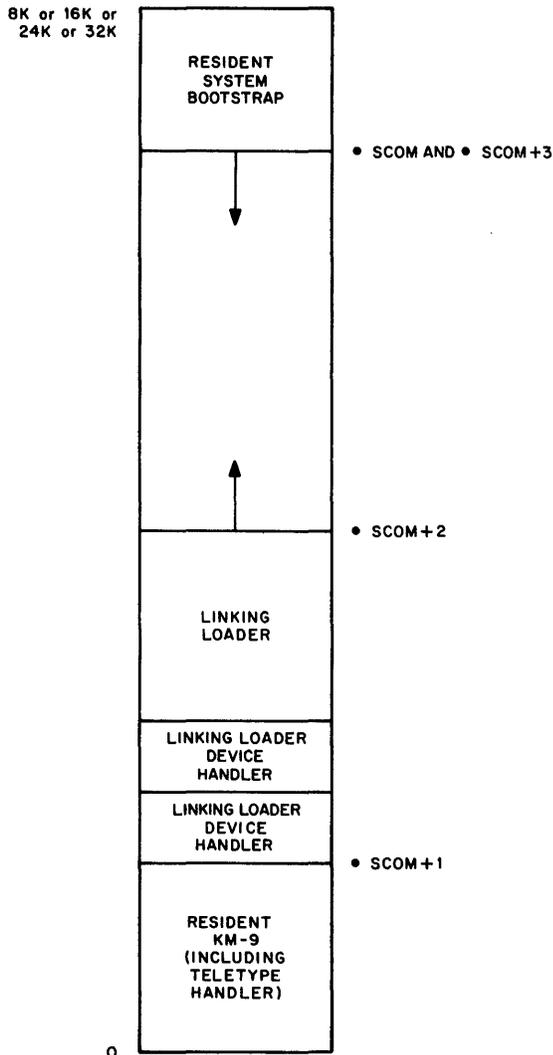
Phase 1

```
8K or 16K or
24K or 32K
             ┌──────────────┐
             │   RESIDENT   │
             │    SYSTEM    │
             │  BOOTSTRAP   │
             ├──────────────┤  • SCOM AND • SCOM+3
             │      │       │
             │      ▼       │
             │              │
             │              │
             │              │
             │      ▲       │
             │      │       │
             ├──────────────┤  • SCOM+2
             │              │
             │   LINKING    │
             │   LOADER     │
             │              │
             ├──────────────┤
             │LINKING LOADER│
             │   DEVICE     │
             │   HANDLER    │
             ├──────────────┤
             │LINKING LOADER│
             │   DEVICE     │
             │   HANDLER    │
             ├──────────────┤  • SCOM+1
             │   RESIDENT   │
             │    KM-9      │
             │ (INCLUDING   │
             │  TELETYPE    │
             │  HANDLER)    │
   0         └──────────────┘
```

The System Loader learns which I/O handlers are required by the Linking Loader, loads them relocatably and then loads the Linking Loader relocatably.
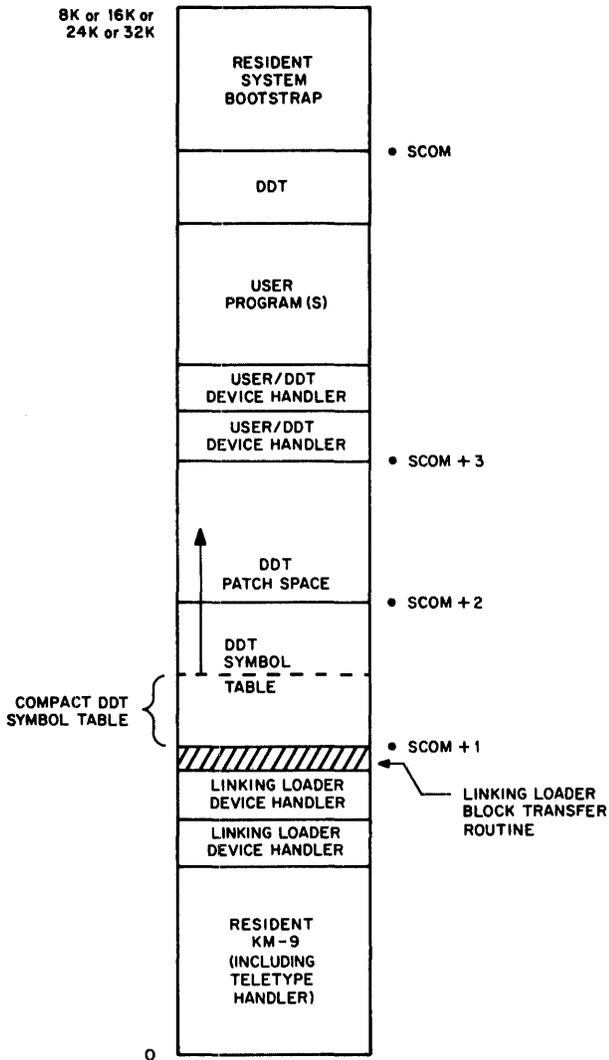
The Linking Loader, during loading of user programs down from .SCOM+3 builds the loader (GLOBAL) and DDT (if DDT) symbol tables up from .SCOM+2.

If a DDT load, the Linking Loader just prior to giving control to DDT moves the DDT symbol table down in core so that it overlays all of the Linking Loader except for the small routine that makes the block transfer.

The Linking Loader will not load a device handler that is already in core for its own use.

Phase 2B (DDT or DDT NS)

8K or 16K or 24K or 32K

RESIDENT SYSTEM BOOTSTRAP
DDT
USER PROGRAM (S)
USER/DDT DEVICE HANDLER
USER/DDT DEVICE HANDLER
DDT PATCH SPACE
DDT SYMBOL TABLE
LINKING LOADER DEVICE HANDLER
LINKING LOADER DEVICE HANDLER
RESIDENT KM-9 (INCLUDING TELETYPE HANDLER)

● SCOM

● SCOM + 3

● SCOM + 2

● SCOM + 1

COMPACT DDT SYMBOL TABLE

LINKING LOADER BLOCK TRANSFER ROUTINE

0

.EXIT from the user program causes the system bootstrap to re-initialize the Keyboard Monitor.

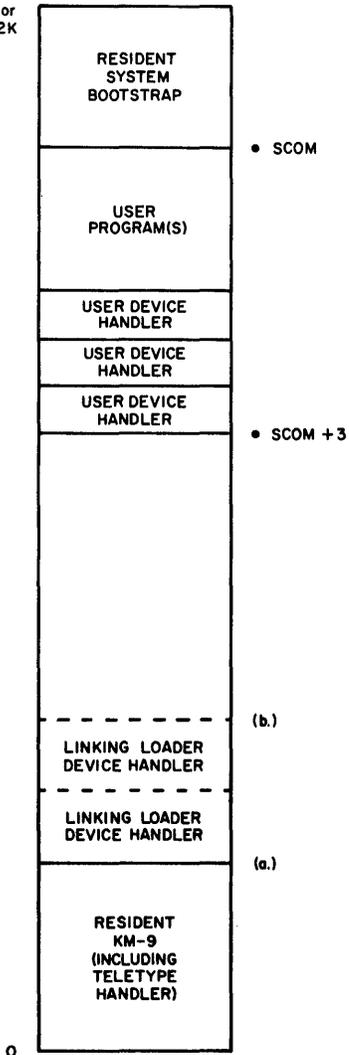If a DDTNS load, no DDT symbol table is built.

Non BLOCK DATA COMMON (FORTRAN IV or MACRO-9 output) may make use of core as low as the compact DDT symbol table (DDT only retains certain symbol table entries). However, the user must be careful about placing patches.

The Linking Loader device handlers would have been used to satisfy user device requests.

Phase 2 (Not DDT or DDTNS)

```
8K or 16K or          ┌─────────────────┐
24K or 32K            │    RESIDENT     │
                      │     SYSTEM      │
                      │   BOOTSTRAP     │
                      │                 │
                      ├─────────────────┤ ● SCOM
                      │                 │
                      │      USER       │
                      │   PROGRAM(S)    │
                      │                 │
                      ├─────────────────┤
                      │   USER DEVICE   │
                      │     HANDLER     │
                      ├─────────────────┤
                      │   USER DEVICE   │
                      │     HANDLER     │
                      ├─────────────────┤
                      │   USER DEVICE   │
                      │     HANDLER     │
                      ├─────────────────┤ ● SCOM + 3
                      │                 │
                      │                 │
                      │                 │
                      │                 │
                      │                 │
                      │                 │
                      ├─ ─ ─ ─ ─ ─ ─ ─ ─┤ (b.)
                      │ LINKING LOADER  │
                      │ DEVICE HANDLER  │
                      ├─ ─ ─ ─ ─ ─ ─ ─ ─┤
                      │ LINKING LOADER  │
                      │ DEVICE HANDLER  │
                      ├─────────────────┤ (a.)
                      │                 │
                      │    RESIDENT     │
                      │      KM-9       │
                      │   (INCLUDING    │
                      │    TELETYPE     │
                      │    HANDLER)     │
     0                └─────────────────┘
```

.EXIT from the user program causes the
system bootstrap to re-initialize the
Keyboard Monitor.

.SCOM+1 and .SCOM+2 point to one of
two places.

    (a) If the user program did not have any
        device handlers in common with the
        Linking Loader.

    (b) If the user program did have at least
        one device handler in common with
        the Linking Loader.

Non BLOCK DATA COMMON (FORTRAN IV
or MACRO-9 output) may make use of core as
low .SCOM+2

APPENDIX 1
SYMBOL CONCATENATION - RADIX 50_8 FORMAT

Radix 50_8 is a technique used by the MACRO-9 Assembler and the FORTRAN IV Compiler to condense the binary representation of symbolic names in symbol tables. Three characters plus two symbol classification bits are contained in each 18-bit word. A symbol is defined as a string of one to six characters, i.e.,

$$C_1 \; C_2 \; C_3 \; C_4 \; C_5 \; C_6$$

where C_i is defined as

Character	6-bit octal code
Space	00
A	01
↓	↓
Z	32
%	33
.	34
0	35
↓	↓
9	46
#	47

The symbol is concatenated as follows:

Word 1 $\quad ((C_1 * 50_8) + C_2) 50_8 + C_3$

Word 2 $\quad ((C_4 * 50_8) + C_5) 50_8 + C_6$

For example: The symbol SYMNAM would be entered in the Loader's symbol table as:

Word 1 $\quad ((23_8 * 50_8) + 31_8) 50_8 + 15_8 \; = \; 475265*$

Word 2 $\quad ((16_8 * 50_8) + 1) 50_8 + 15_8 \; = \; 053665$

---

*The sign bit of WORD 1 is set to 1 to indicate that this symbol consists of more than 3 characters and that the WORD 2 is necessary.

APPENDIX 2
LOADER SYMBOL TABLE

Common Block Name

	0 2 3	17
1	ID	Block Size
2	Name (2A)	
3	Symtab address of last entry in chain	
4	Block Definition	

ID = 7 when not defined

ID = 3 when defined

0 if no entries

0 if not defined

"Name" may require 2 words.

Common Name

	0 2 3	17
1	ID	Symtab Chain Address
2	1	TV Address
3	Relative Address in Block	

0 if last entry in chain
ID = 4
B0 = 1 for easy entry
update

If associated COMMON block was defined when code 14 is encountered, no entry is needed in the symbol table.

Virtual Global (internal)

	0 3	17
1	ID	Definition
2	Name (2A)	

Definition (Virtual) = Absolute Address of last TV in chain   Virtual   ID = 1

Definition (Internal) = Absolute Address of Symbol   Internal   ID = 5

"Name" may require 2 words.

Internal Names

	0 3	17
1	ID	Definition
2	Name (2A)	

"Name" may require 2 words.

ID = 0

If Program Name
ID = 6

Only entered
into the symbol
table during
DDT loads.

7-to-9 CONVERTER

CONTENTS

1.   INTRODUCTION

Source programs written for the PDP-7 Assembler in ASCII (or the PDP-9 BASIC Software System Symbolic Assembler) may be converted to the source language and statement format of the PDP-9 ADVANCED Software System Assembler, MACRO-9, by the 7-to-9 Converter program. FIODEC is not accepted by the Converter. It is assumed that the reader is familiar with both assembler formats.

The Converter operates in the PDP-9 ADVANCED Software System environments, with either the I/O Monitor (paper tape system) or the Keyboard Monitor (DECtape or other mass storage systems).

Basically, this program converts statements in the input PDP-7 program to equivalent MACRO-9 statements. Some PDP-7 Assembler pseudo-ops cannot be translated because MACRO-9 does not perform a comparable function. These are not changed by the converter, but will be flagged as undefined symbols when assembled by MACRO-9. PDP-7 pseudo-ops which cannot be converted are listed below.

ANALEX	FIODEC	NOSYMBOLS	SYMBOLS
BAR	FIX	PUNCH	TELETYPE
CHAR	FLEX	PUNDEF	TEXT
EXPUNGE	NOINPUT		VARIABLES

Since MACRO-9 does not allow multiword variables, the dollar sign ($) should not appear in the input source program.

2.   CONVERTER FUNCTIONS

The converter performs the following functions.

a. Removes commas from tags (or labels).

b. Removes Location Counter Settings. For example, 100/ is normally exactly translated as .LOC 100, but the user may specify, in the command string that the Location Counter setting be removed completely.

c. If another statement follows on the same line, the converter inserts the semicolon delimiter as required in the MACRO-9 statement format.

d. Inserts plus signs where needed LAC   A   5 is translated to LAC A+5.

e. Changes the indirect address indicator from I to *, as LAC   I   A to LAC* A.

Normally, the converter does not produce a printed listing, and terminates programs with an .END statement. The user may make command string entries, however, to request the following functions.

a. A printed listing

b. Insert the .ABS pseudo-op

c. Remove Location Counter settings

1

    d. Terminate physical segments with .EOT instead of .END

    e. Multiple inputs

3.   OPERATING INSTRUCTIONS

3.1   Command String Format

After the converter types,

        7-TO-9 CONVERTER

        >

the user types the command string in the following format,

optional	input name	output name	terminator
> L, A, R, E, Tn ←	file1	, file2	↓ (or ALT mode)

where, if typed,

    L   Requests a printed output listing

    A   Insert .ABS

    R   Remove Location Counter settings

    E   Terminate with .EOT

    T   Multiple input, followed by n

    n   Number of inputs

The reverse arrow must follow the optional function entries, or start the command string if no optional entries are made.

    file1   Input program name, if different from output, otherwise it is omitted.

    file2   Name of the program to be output. May be used to rename the program.

If the command string is terminated by a carriage return, on completion of conversion, control returns to the 7-to-9 Converter to convert another program. If terminated by ALT mode, control returns to Monitor (if in Keyboard Monitor environment).

Optional entries may appear in any order, separated by commas, and terminated by the reverse arrow. Rubouts may be used to delete any unwanted characters prior to typing the command string terminator. If an error is detected, the Converter types,

        COMMAND STRING ERROR

        >

and the user may type the corrected command string.

The following command strings are valid and correct,

>L,E,T3 ← NAME7, NAME9)

>T2,R,A ← SEVEN, NINE)

> ← SAME (ALT)

In the last example, there will be no listing, no .ABS insertion, Location Counter settings will be converted to .LOCs, .END will terminate, and only one input will be allowed. Both the input and output program are named SAME, and upon conversion, control returns to Monitor.

3.2 With I/O Monitor

In the paper tape only environment, to load the 7-to-9 Converter, place the CONV tape in the reader, set the address switches to 17720 of the highest memory bank, depress the I/O RESET switch, and then depresses the hardware READIN switch. When the Converter is ready to receive a command string, it types,

7-TO-9 CONVERTER

>

3.3 With Keyboard Monitor

The Converter is called by typing CONV, after the Keyboard Monitor has typed $. When ready to receive a command string, the Converter types,

7-TO-9 CONVERTER

>

3.3.1 Device Assignments - The Converter assumes that the input is assigned .DAT slot -14, the output is assigned .DAT slot -15, and the listing device is assigned -12. The user may check the current device assignments by typing $ REQUEST CONV, and he may use the ASSIGN command to modify the assignments if desired.

4. USING THE CONVERTER

It is normally expected that some editing will be necessary to the output of the converter. The converter performs the tedious operation of adjusting statement format from that of the Basic Assembler to that of MACRO-9. If any of the pseudo-ops listed in Section 1 are used, editing must be done before the converted program will assemble correctly.

If the converted program is to be in relocatable form and run in the PDP-9 Monitor environment (I/O or Keyboard), the input/output procedures must be revised to utilize the IOPS routines.

Any device IOT instructions which are to be kept in the converted programs must be defined by statements such as TSF = 700401. The Editor may be used to insert these definitions at the front of the converter output.

The procedure to convert a program for assembly in the absolute (.ABS) mode is as follows.

    a.   Be sure the source tape is punched in ASCII.

    b.   Run the converter (CONV-9), using option A to place .ABS on the converted program.

    c.   An assembly may be run to locate any illegal codes not corrected by the converter.

    d.   Edit

        1.   revise coding to remove pseudo-ops which MACRO-9 cannot handle

        2.   define device IOT instructions

    e.   Assemble with MACRO-9.

The procedure to convert a program for assembly in the relocatable mode, for running in the monitor environment, is as follows:

    a.   Be sure the source tape is punched in ASCII.

    b.   Run the converter (CONV-9). Do not use option A.

    c.   An assembly may be run to locate any illegal codes not corrected by the converter.

    d.   Edit

        1.   revise coding to remove pseudo-ops which MACRO-9 cannot handle.

        2.   revise input/output procedures to utilize the IOPS routines.

5.   ASSEMBLE WITH MACRO-9.

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively, we need user feedback: your critical evaluation of this manual and the DEC products described.

Please comment on this publication. For example, in your judgment, is it complete, accurate, well-organized, well-written, usable, etc? _____

_____

_____

_____

_____

Did you find this manual easy to use? _____

_____

_____

What is the most serious fault in this manual? _____

_____

_____

_____

What single feature did you like best in this manual? _____

_____

_____

_____

Did you find errors in this manual? Please describe. _____

_____

_____

_____

Please describe your position. _____

Name _____ Organization _____

Street _____ State _____ Zip _____