PAL-D
DISK ASSEMBLER
PROGRAMMER'S REFERENCE MANUAL

For additional copies of this document, order No. DEC-D8-ASAB-D from Program Library,
Digital Equipment Corporation, Maynard, Mass. 01754 Price: $1.50

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing April 1968
2nd Printing June 1968
3rd Printing March 1969
4th Printing (Rev) September 1969

Your attention is invited to the last two pages of this manual. The Reader's Comments page, when filled in and returned, is beneficial to both you and DEC. All comments received are considered when documenting subsequent manuals, and when assistance is required, a knowledgeable DEC representative will contact you. The Software Information page offers you a means of keeping up-to-date with DEC's software.

Documents Referenced (available from DEC's Program Library):

Introduction to Programming, C-18

Disk Monitor System, Programmer's Reference Manual, DEC-D8-SDAB-D

Time-Sharing System User's Guide, DEC-T8-MRFB-D

TSS/8 System Manager's Guide, DEC-T8-MBZA-D

PREFACE

PAL-D, one of the PDP-8 family assembly programs, is designed for use on any PDP-8 family computer with disk or DECtape secondary storage. It is loaded, optionally stored on disk as a permanently resident utility program and reproduced in core image as required, under control of the PDP-8/I Disk Monitor, or the TSS/8 Time-Sharing Monitor.

PAL-D produces a binary coded object program after two passes of the symbolic coded source program. An optional third pass produces a listing of the source program and the assembler-generated binary code expressed as four-digit octal values.

Along with the standard assembly functions PAL-D offers double precision integers, floating point constants, arithmetic and Boolean operators, literals, text facilities and automatic off-page linkage generation as standard features.

It is assumed that the reader is familiar with assembly language programming. For an elementary approach to this type of programming, we recommend DEC's publication, No. C-18, "Introduction to Programming" available from the Program Library, Digital Equipment Corporation, Maynard, Massachusetts.

CONTENTS

CONTENTS (Cont)

CHAPTER 1
INTRODUCTION

PAL-D, the acronym for Program Assembly Language for the Disk, is the symbolic assembly program designed primarily for the 4K PDP-8 family of computers with disk or DECtape secondary storage operated in either stand-alone or time-shared mode.

The PAL-D Assembler makes machine language programming easier, faster, and more efficient. Basically, the Assembler processes the programmer's source program statements by translating mnemonic operation codes to the binary codes needed in machine instructions, relating symbols to numeric values, assigning absolute core addresses for program instructions and data, and preparing an output listing of the program, which includes notification of any errors detected during the assembly process.

The PAL-D Assembly language is the same under both the Disk Monitor System and the TSS/8 Monitor (time-sharing) System. The assembly system includes the disk version of the Symbolic Tape Editor for altering or editing the source language tape, the Disk Debugging technique for debugging the object program by communicating with it in the source language, and various other utility programs.

PAL-D requires the minimum configuration for disk or DECtape systems (see Disk Monitor System, DEC-D8-SDAB-D) or time sharing systems (see Time-Sharing System User's Guide, DEC-T8-MRFB-D), and additionally can utilize the high-speed reader/punch and up to three additional DS32 disk units.

1.1 PAL-D LANGUAGE

The PAL-D Assembler is compatible with the PAL III Assembler. However, PAL-D has the following additional features.

Operators	Symbols and integers may be combined by using the operators +Addition & Boolean AND -Subtraction ! Boolean Inclusive OR
Literals	Symbolic or integer literals (constants) are automatically assigned.
Text Facility	Text facilities exist for single characters and blocks of text.
Indirect Linkage Generation	Indirect links are automatically generated for off-page referencing.

1.2 SYNTAX

Programs processed under PAL-D are written using USA SCII characters. Appendix A contains a complete list of these characters with their octal code equivalents.

1.2.1 Legal Characters

The following characters are acceptable to PAL-D.

a. The alphabetic characters
   ABCD...XYZ

b. The numeric characters
   0 1 2 3 4 5 6 7 8 9

c. The special characters

␣	Space	Separates symbols and numbers (see Section 1.5.1)
+	Plus	Combines symbols or numbers (add)
-	Minus	Combines symbols or numbers (subtract)
!	Exclamation Mark	Combines symbols or numbers (inclusive OR)
↵	Carriage Return	Terminates a line
→|	Tabulation	Formats symbols or numbers or source tape output
,	Comma	Assigns symbolic address
=	Equal Sign	Direct assignment of symbol values
;	Semicolon	Terminates coding line (will not terminate comments)
$	Dollar Sign	Indicates end of pass
*	Asterisk	Sets current location counter; redefines origin
.	Point (Period)	Has value equal to current location counter
/	Slash	Indicates start of comment
&	Ampersand	Combines symbols or numbers (AND)
"	Quote	Generates USA SCII constant
( )	Parentheses	Defines literal on current page
[ ]	Brackets	Defines page 0 literal

d. Ignored characters

Form-Feed	Indicates the end of a logical page of source program
Blank Tape	Used for leader/trailer
Code 200	Used for leader/trailer
Rubout	Follows tabulation characters for timing purposes
Line-Feed	Follows carriage return and causes teleprinter paper to roll upward one line

Since certain characters are invisible (i.e., nonprinting), the following symbols are used throughout this manual to represent their presence.

␣	Space
→|	Tabulation
↵	Carriage Return

1.2.2 Illegal Characters

All characters other than those listed above are illegal when not in a comment or TEXT field and, being illegal, their occurrence causes the error message IC (Illegal Character) to be printed by PAL-D.

1.2.3 Format Effectors

Tabulations are usually used in the body of a source program to provide a neat page; they can separate fields from one another, as between a statement and a comment. For example, a line written

GO, TAD TOTAL/MAIN LOOP

is much easier to read if tabs are inserted to form

GO, →| TAD TOTAL →| /MAIN LOOP

Either the ";" (semicolon) or "↵ " (carriage return-line feed) character may be used as a statement terminator. The semicolon is considered identical to carriage return-line feed except that it will not terminate a comment. Example:

TAD A /THIS IS A COMMENT; TAD B↵

The entire expression between the "/" (slash) and ↵ (carriage return) is considered a comment.

The semicolon also allows the programmer to place several lines of coding on a single line. If, for example, he wishes to write a sequence of instructions to rotate the contents of the accumulator and link six places to the right, it might look like

```
...
RTR↵
RTR↵
RTR↵
...
```

The programmer may place all three RTRs on a single line by separating them with the special character ";" and terminating the line with a carriage return. The above sequence of instructions can then be written

RTR; RTR; RTR↵

This format is particularly useful when setting aside a section of data storage for a list. For example, a 12-word list could be reserved by specifying the following format.

```
LIST,   0;  0;  0;  0;  0;  0↵
        0;  0;  0;  0;  0;  0↵
```

A neat printout (or program listing) makes subsequent editing, debugging, and interpretation much easier than when the coding is laid out in a haphazard fashion.

1.3 STATEMENTS

PAL-D source programs are usually prepared on a Teletype, with the aid of the Editor, as a sequence of statements. Each statement is written on a single line and is terminated by a carriage return-line feed sequence. PAL-D statements are virtually format free; that is, elements of a statement are not placed in numbered columns with rigidly controlled spacing between elements, as in punched-card oriented assemblers.

There are four types of elements in a PAL-D statement which are identified by the order of appearance in the statement, and by the separating, or delimiting, character which follows or precedes the element.

Statements are written in the general form

label, operator operand/comment

The Assembler interprets and processes these statements, generating one or more binary instructions or data words, or performing an assembly process. A statement must contain at least one of these elements and may contain all four types.

1.3.1 Labels

A label is the symbolic name created by the source programmer to identify the position of the statement in the program. If present, the label is written first in a statement and terminated by a comma.

1.3.2 Operators

An operator may be one of the mnemonic machine instruction codes (see Appendix B), or a pseudo-operation (pseudo-op) code which directs assembly processing. The assembly pseudo-op codes are described in Chapter 2. Operators are terminated with a space if an operand follows or with a semicolon, slash, or carriage return.

1.3.3 Operands

Operands are usually the symbolic address of the data to be accessed when an instruction is executed, or the input data or arguments of a pseudo-op. In each case, interpretation of operands in a statement depends on the statement operator. Operands are terminated by a semicolon, a slash if a comment follows, or a carriage return-line feed.

1.3.4 Comments

The programmer may add notes to a statement following a slash mark. Such comments do not affect assembly processing or program execution, but are useful in the program listing for later analysis or debugging.

1.4 Symbols

The programmer may create symbols to use as statement labels, as operators, and as operands. A symbol is a string of one or more alphanumeric characters delimited by a punctuation character. A symbol contains from one to six characters from the set of 26 alphabetic characters and ten digits 0 through 9; however, the first character must be alphabetic.

1.4.1 Symbol Distinction

The PAL-D Assembler makes a distinction between the types of symbols it is processing. These types are

a. Permanent symbols

JMS — a symbol whose value of 4000 (octal) is taken from PAL-D's permanent operation code symbol table.

b. User-defined symbols

HERE — a user-defined symbol; when used as a symbolic address tag, its value is the address of the statement it tags (this value is assigned by PAL-D).

1.4.1.1 Permanent Symbols - PAL-D has in its permanent symbol table definitions of its operation codes, operate commands, and many input-output transfer (IOT) microinstructions (see Appendix B). PAL-D's permanent symbols may be used without prior definition by the user.

1.4.1.2 User-Defined Symbols - User-defined symbols are composed according to the following rules.

a. The characters must be alphabetic (A-Z) or numeric (0-9).

b. The first character must be alphabetic.

c. Only the first six characters of any symbol are meaningful to PAL-D; the remainder, if any, are ignored.

Note that because of the third rule above, a symbol such as INTEGER would be interpreted as INTEGE since the seventh character is ignored. Remember, if symbols of more than six characters are used, the programmer must avoid defining two apparently different symbols whose first characters are identical. For example, the two symbols GEORGE1 and GEORGE2 differ only in the seventh character, thus the Assembler treats them as being the same symbol, GEORGE.

When the symbol following the space is a user-defined symbol, the space acts as an address field delimiter. Example:

```
     *2117↵
A,   CLA↵
      .
      .
      .
     JMP␣A↵
```

where A is user-defined symbol with the value 2117. The expression JMP A is evaluated as follows.

```
      JMP          101  000  000  000 (binary representation of permanent symbol JMP)
Address A          000  011  001  111 (binary representation of address A)
```

The operation codes (op codes) are inclusively ORed to form

JMP A 101 011 001 111

or written more concisely in octal as 5317.

1.4.2 Symbolic Addresses

A symbol used as a label to specify a symbolic address must appear first in the statement and must be immediately followed by a comma. When used in this way, a symbol is said to be defined. A defined symbol can reference an instruction or data word at any point in the program. A symbol can be defined as a label only once. If a programmer attempts to define the same symbol as a label again, the second or successive attempt is ignored and an error is indicated. The Assembler recognizes only the first definition. These are legal symbolic addresses:

```
ADDR,
TOTAL,
SUM,
```

The following symbolic addresses are illegal:

```
7ABC,      (first character must be alphabetic)
LAB—,      (comma must immediately follow label)
```

1.4.3 Symbolic Operators

Symbols used as operators must be predefined by the Assembler or by the programmer. If a statement has no label, the operator may appear first in the statement, and must be terminated by a space, tab, semicolon, or carriage return. The following are examples of legal operators:

```
TAD        (a mnemonic machine instruction operator)
PAGE       (an Assembler pseudo-op)
ZIP        (legal only if defined by the user)
```

1.4.4 Symbolic Operands

Symbols used as operands must have a value defined by the user. These may be symbolic references to previously defined labels where the arguments to be used by this instruction are to be found, or the values of symbolic operands may be constants or character strings.

```
TOTAL,          TAD AC1 + TAG
```

The first operand, AC1, specifies an accumulator register, determined by the value given to the symbol AC1 by the user. The second operand references a memory location whose name or symbolic address is TAG.

1.4.5 Symbol Tables

The Assembler processes symbols in source program statements by referencing its symbol tables which contain all defined symbols along with the binary value assigned to each symbol.

Initially, the Assembler's permanent symbol table contains the mnemonic op codes of the machine instructions and the Assembler pseudo-op codes, as listed in Appendix B. As the source program is processed, symbols defined in the source program are added to the user's symbol table.

1.4.5.1 Direct Assignment Statements - The programmer inserts new symbols with their assigned values directly into the symbol table by using a direct assignment statement of the form

symbol = value

where the value may be a number or expression. For example,

```
ALPHA=5
BETA=17
```

A direct assignment statement may also be used to give a new symbol the same value as a previously defined symbol.

```
BETA=17
GAMMA=BETA
```

The new symbol, GAMMA, is entered into the user's symbol table with the value 17.

The value assigned to a symbol may be changed.

```
ALPHA=7
```

changes the value assigned to the first example from 5 to 7.

The user may also define symbols by use of the comma. When the first symbol of a statement is terminated by a comma, it is assigned a value equal to the current location counter (CLC). For example,

```
          *100          /set CLC (origin) to 100↵
TAG,      CLA↵
          JMP A↵
B,        O↵
A,        DCA B↵
          ...
```

The symbol TAG is assigned a value of 0100, the symbol B a value of 0102, and the symbol A a value of 0103.

Direct assignment statements do not generate instructions or data in the object program. These statements are used to assign values so that symbols can be conveniently used in other statements.

1.5 NUMBERS

Any sequence of numbers delimited by a punctuation character is interpreted numerically by PAL-D.

1
12
4372

The radix control pseudo-operators (pseudo-ops) indicate to the Assembler the radix to be used in number interpretation (see Chapter 2). The pseudo-op DECIMAL indicates that all numbers are to be interpreted as decimal until the next occurrence of the pseudo-op OCTAL. The pseudo-op OCTAL indicates that all numbers are to be interpreted as octal until the next occurrence of the pseudo-op DECIMAL.

The radix is initially set to octal and remains octal unless otherwise specified.

1.5.1 Arithmetic and Logical Operators

The arithmetic and logical operators are:

+	Plus	2s complement addition (modulo 4096)
-	Minus	2s complement subtraction (modulo 4096)
!	Exclamation Mark	Boolean inclusive OR (union)
&	Ampersand	Boolean AND (intersection)
␣	Space	Interpreted as inclusive OR when used to separate two symbolic operators. Example: TAG, CLA ␣CLL ↵

1.5.2 Evaluating Expressions

Symbols and numbers (exclusive of pseudo-op symbols) may be combined by using the arithmetic and logical operators to form expressions. Expressions are evaluated from left to right. Example:

	A	B	A+B	A-B	A! B	A&B
Value	0002	0003	0005	7777	0003	0002
Value	0007	0005	0014	0002	0007	0005
Value	0700	0007	0707	0671	0707	0000

1.6 ADDRESS ASSIGNMENTS

The PAL-D Assembler sets the origin, or starting address, of the source program to absolute location (address) 0200 unless the origin is specified by the programmer. As source statements are processed, PAL-D assigns consecutive memory addresses to the instructions and data words of the object program. This is done by incrementing the location counter each time a memory location is assigned. A statement which

generates a single object program storage word increments the location counter by one. Another statement may generate six storage words, thus incrementing the location counter by six.

Direct assignment statements and some Assembler pseudo-ops do not generate storage words and therefore do not affect the location counter.

1.6.1 Current Address Indicator

The special character . (point or period) always has a value equal to the value of the current location counter. It may be used as any integer or symbol (except to the left of an equal sign). Example:

```
*200↵
JMP .+2↵
```

is equivalent to JMP 0202. Also,

```
*300↵
.+2400↵
```

will produce in location 0300 the quantity 2700. Consider

```
*2200↵
CALL=JMS I .↵
0027↵
```

The second line, CALL = JMS I ., does not increment the current location counter, therefore, 0027 is placed in location 2200 and CALL is placed in the user's symbol table with an associated value of 4600 (the octal equivalent of JMS I.).

1.6.2 Indirect Addressing

When the character I appears in a statement between a memory reference instruction and an operand, the operand becomes the address containing the address of the statement to be executed. Consider

```
TAD 40
```

which is a direct address statement, where 40 is interpreted as the address containing the quantity to be added to the accumulator. Thus, if address 40 contains 0432, then 0432 is added to the accumulator. Now consider

```
TAD I 40
```

which is an indirect address statement, where 40 is interpreted as the address of the address containing the quantity to be added to the accumulator. Thus, if address 40 contains 432, and address 432 contains 456, then 456 is added to the accumulator.

When a reference is made to an address not on the same page as the reference, PAL-D sets the indirect bit (bit 3) of the machine instruction, generating an indirect address linkage to the off-page reference (see Paging and Off-Page Referencing, Sections 1.7.1.1 and 1.7.1.2).

In the case of several off-page references to the same address, the indirect address linkage will be generated only once.

Example:

```
        *2117↵
A,      CLA ↵
        .
        .
        .
        *2600↵
        TAD  A
        .
        .
        .
        DCA  A
```

The space preceding the user-defined symbol A acts as an address field delimiter. PAL-D will recognize that the address tag A is not on the current page (in this case 2600-2777) and will generate a link to it in the following manner. In location 2600, PAL-D will place the word

1777 (octal equivalent of TAD I 2777)

and in location 2777 (the last location on the current page) the word 2117 (the actual address of A) will be placed. When it sees the second reference to A it will use the previous link word rather than creating a new one.

PAL-D will recognize and generate an indirect address linkage only when the address referenced is to a location on another page, not the current page. The programmer must use the character I to indicate an explicit indirect address when indirectly addressing to a location on the current page.

PAL-D cannot generate a link for an instruction that is already specified as being an indirect address. In this case, PAL-D will type the error message II (Illegal Indirect); the error message is ignored and assembly is continued.

1.6.3 Autoindexing

Interpage references are often necessary for obtaining operands when processing large amounts of data. The PDP-8 computers have facilities to ease the addressing of this data. When absolute locations 10 to 17 (octal) are indirectly addressed, the content of the location is incremented before it is used as an address and the incremented number is left in the location. This allows the programmer to address consecutive memory locations using a minimum of statements.

It must be remembered that initially these locations (10 to 17) must be set to one less than the first desired address. Because of their characteristics, these locations are called autoindex registers. No incrementation takes place when locations 10 to 17 are addressed directly.

Example:

```
Statement is in location 500
Data is on the page starting at 5000
Autoindexing register 10 is used for addressing
```

```
0476  1377  TAD (5000-1)          / set up auto
0477  3010  DCA 10               / index with 4777
0500  1410  TAD I 10             / C(10) is incremented to 5000 before
  :     :                          use as address
0577  4777                       / literal generated by PAL-D
```

When the statement in location 500 is executed, the content of location 10 will be incremented to 5000 and the content of location 5000 will be added to the content of the accumulator. If the instruction TAD I 10 is re-executed, the content of location 5001 is added to the content of the accumulator, and so on.

1.6.4 Literals

Symbolic and integer literals (constants) may be defined as shown below.

```
. . .
CLA ↵
TAD (2) ↵
DCA INDEX ↵
. . .
```

Operator and operand must always be separated with a space.

The left parenthesis is a signal to the Assembler that the integer following is to be assigned a location in the table at the top of the current page. This is the same table in which the indirect address linkages are stored. In the above example, the quantity 2 is stored in the first free location in a list beginning at the top of the current page (relative address 177), and the statement in which it appears is encoded with an address referring to that location.

A literal is assigned to storage the first time it is encountered; subsequent references will be to the same location.

If the programmer wishes to assign literals to page 0 rather than the current page, he must use square brackets, [ ], in place of parentheses. Whether using parentheses or square brackets, the right or closing member is optional and may always be replaced with a carriage return.

```
TAD (777 ↵
```

1.6.4.1 Nesting - Literals may be nested as shown below.

```
*200 ↵
TAD (TAD (30 ↵
```

will generate

```
0200    1276
 ...     ...
0376    1377        (literals assigned to locations
0377    0030        0377 and 0376; top of current page)
```

This type of nesting may be carried to many levels.

Literals are stored on each page starting at relative address 177 (only 127_{10} or 177_8 literals may be placed on page 0). If literals are being generated for some nonzero page and then the origin is set to another page, the current page literal buffer is punched out during pass 2. If the origin is reset to the previously used page, the same literal will be generated if used again.

If a single character is preceded by a quote ("), the 8-bit value of the USA SCII code for that character is inserted instead of taking the letter as a symbol.

Example:

```
CLA↵
TAD ("A↵
...
```

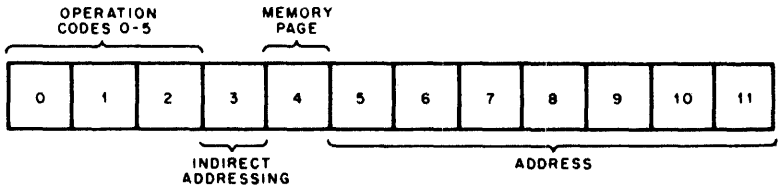
will place the constant 0301 in the accumulator.

1.7 INSTRUCTIONS

There are two basic groups of instructions: memory reference and augmented. Memory reference instructions require an operand; augmented instructions do not require an operand.

1.7.1 Memory Reference Instructions

In PDP-8 computers, some instructions require a reference to memory. They are appropriately designated memory reference instructions, and take the following format.



Memory Reference Instruction Bit Assignments

Bits 0 through 2 contain the operation code of the instruction to be performed (AND, TAD, DCA, JMS, or or JMP). Bit 3 tells the computer if the instruction is indirect, that is, if the address of the instruction specifies the location of the operand, or if it specifies the location of the address of the operand. Bit 4 tells the computer if the instruction is referencing the current page or page zero. This leaves bits 5 through 11 (7 bits) to specify an address. In these 7 bits, 200 octal or 128 decimal locations may be specified; the page bit increases accessible locations to 400 octal or 256 decimal.

The address field of a memory reference instruction may be any valid expression.

Example:

```
A=270↵
*200↵
TAD  A-20↵
```

produces, in location 200, the word

1250

which in binary is 001 010 101 000

which is also TAD 250.

1.7.1.1 Paging - To ease the programmer's addressing problems, a convention has been defined that divides memory into sectors called pages. Each page contain 200 octal locations (128 decimal) numbered 0 to 177 (octal) on that page. There are 40 octal or 32 decimal pages numbered 0 to 37 (octal). Some examples of page numbers and the absolute and relative locations (addresses) are shown below. It must be borne in mind, however, that there is no physical separation of pages in memory.

Page	Absolute Address	Relative Address
0	0 - 177	0 - 177
1	200 - 377	0 - 177
2	400 - 577	0 - 177
36	7400 - 7577	0 - 177
37	7600 - 7777	0 - 177

The following table offers a comparison of specific absolute and relative addresses on the same page.

Page	Absolute Address	Relative Address
0	10	10
3	617	17
12	2577	177
31	6255	55
37	7777	177

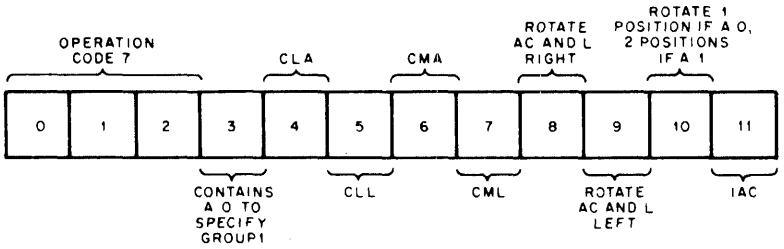
Since only seven bits are necessary to address 200 octal locations, bits 5 to 11 are reserved for this function.

1.7.1.2 Off-Page Referencing - The page on which an absolute address is contained can be determined from bit 4 of the instruction. If bit 4 is a 0, the address refers to a location on page 0; if bit 4 is a 1, the address refers to a location on the current (same) page, that is, the same memory page as the instruction.

1.7.2 Augmented Instructions

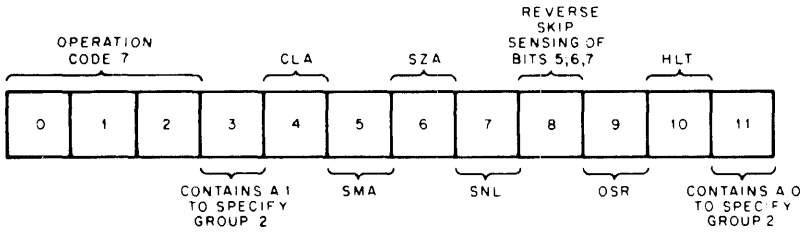
Augmented instructions are divided into two groups: operate and input-output transfer microinstructions.

1.7.2.1 Operate Microinstructions - Within the operate group there are two groups of microinstructions. Group 1 microinstructions are principally for clear, complement, rotate, and increment operations and are designated by the presence of a 0 in bit 3 of the machine instruction word. (See Appendix B.)



Group 1 Operate Microinstruction Bit Assignments

Group 2 microinstructions are used principally in checking the content of the accumulator and link and, based on the check, continuing to or skipping the next statement. Group 2 microinstructions are identified by the presence of a 1 in bit 3 and a 0 in bit 11 of the machine instruction word (See Appendix B).



Group 2 Operate Microinstruction Bit Assignments

Group 1 and group 2 microinstructions can not be combined because bit 3 determines only one or the other.

Within Group 2, there are two groups of skip instructions. They may be referred to as the OR group and the AND group.

OR Group	AND Group
SMA	SPA
SZA	SNA
SNL	SZL

The OR group is designated by a 0 in bit 8, the AND group by a 1 in bit 8. OR and AND group instructions cannot be combined because bit 8 determines only one or the other.

If the programmer does combine legal skip instructions, it is important to note the conditions under which a skip may occur.

a. OR Group - If these skips are combined in a statement, the inclusive OR of the conditions determines the skip.

SZA SNL

The next statement is skipped if

the accumulator contains 0000, or
the link is a 1, or
both conditions exist.

b. AND Group - If the skips are combined in a statement, the logical AND of the conditions determines the skip.

SNA SZL

The next statement is skipped only if the accumulator differs from 0000 and the link is 0.

1.7.2.2 Input-Output Transfer Microinstructions - These microinstructions initiate operation of peripheral equipment and effect information transfer between the central processor and the input-output device (s). This is the principal function of the input-output transfer (IOT) microinstructions. Appendix B lists all valid IOT microinstructions, and each is discussed in detail in the User's Handbook.

CHAPTER 2
PSEUDO-OPERATORS

The programmer may use pseudo-operators (pseudo-ops) to direct the Assembler to perform certain tasks or to interpret subsequent coding in a certain manner. Some pseudo-ops generate storage words in the object program, other pseudo-ops direct the Assembler on how to proceed with the assembly. Pseudo-ops are maintained in the Assembler's permanent symbol table.

The function of each PAL-D pseudo-op is described below.

2.1 CURRENT LOCATION COUNTER

The programmer may use the PAGE pseudo-op to reset the current location counter (CLC) to the first location on a specified page.

PAGE — without an argument, the CLC is reset to the first location on the next succeeding page. Thus, if a program is being assembled into page 1 and the programmer wishes to begin the next segment of his program on page 2, he need only insert PAGE, as follows.

```
        JMP  .-7↵     (Last location used on page 1)
    PAGE ↵
        CLA ↵         (First location on page 2)
```

PAGE n — resets the CLC to the first location of page n, where n is an integer, a previously defined symbol, or a symbolic expression. Example:

```
PAGE 2      (sets the CLC to location 400)
PAGE 6      (sets the CLC to location 1400)
```

2.2 EXTENDED MEMORY

When using more than one memory bank, the pseudo-op FIELD instructs the Assembler to output a field setting.

FIELD n — where n is an integer, a previously defined symbol, or a symbolic expression within the range $0 \leq n \leq 7$.

This pseudo-op causes a field setting (binary word) of the form

11 XXX 000 where $000 \leq XXX \leq 111$

to be output on the binary tape during pass 2. This word is interpreted by the Loader, which then begins loading information from the Loader into the new field.

2.3 RADIX CONTROL

Integers used in a source program are usually taken as octal numbers. If, however, the programmer wishes to have certain numbers treated as decimal, he may use the pseudo-op DECIMAL.

DECIMAL	all integers in subsequent coding are taken as decimal until the occurrence of the pseudo-op OCTAL.
OCTAL	resets the radix to its original octal base.

2.4 LISTING CONTROL

During pass 3, a listing of the source program is printed (punched). The programmer may, however, control the output of his pass 3 listing by use of the pseudo-op XLIST.

XLIST	Those portions of the source program enclosed by XLIST will not appear in the pass 3 listing.

2.5 TEXT FACILITY

The pseudo-op TEXT enables the user to represent a character or string of characters in USA SCII code trimmed to six bits and packed two characters to a word. The numerical values generated by TEXT are left-justified in the storage words they occupy, with the unused bits of the last word filled with 0s.

A string of text may be entered by giving the pseudo-op TEXT followed by a space, a delimiting character, a string of text, and the same delimiting character.

Example:

```
TEXT ATEXT STRINGA
```

The first printing character following TEXT is taken as the delimiting character, and the text string is the characters which follow until the delimiting character is again encountered.

If the example above were at location 0200, the pass 3 listing would be as follows.

```
200     2405    TE
201     3024    XT
202     4023    ␣S      (␣ denotes a space)
203     2422    TR
204     1116    IN
205     0700    G
```

NOTE

With TEXT, any printing character may be used as a delimiting character; the delimiting character cannot be used in the text string.

2.6 END OF PROGRAM

The special symbol $ (dollar sign) indicates the end of a program. When the Assembler encounters the $, it terminates the pass.

2.7 END OF FILE

The pseudo-op PAUSE signals the Assembler to stop processing the current input file. The current pass is not terminated, and processing continues when the user types CTRL/P.

When processing a segmented program, the programmer must use the PAUSE pseudo-op as the last statement of each segment (tape or file) to halt processing, giving him time to call (or insert, if paper tape is being used) the succeeding segment of his program.

The PAUSE pseudo-op should be used only at the physical end of a tape or file.

2.8 ALTERING THE SYMBOL TABLE

PAL-D has a permanent symbol table which contains all instructions (symbols and their octal values) required by the Disk Monitor System. They are referred to as PAL-D's basic instructions or symbols, and are listed in Appendix B.

When the symbolic program to be assembled requires instructions not already in the table (e.g., card reader IOT's), the table must be altered to include those instructions. PAL-D has two pseudo-ops that are used to alter the permanent symbol table:

EXPUNGE	deletes the entire permanent symbol table, except pseudo-ops.
FIXTAB	appends symbols to the table for duration of the assembly. All symbols defined before the occurrence of FIXTAB are temporarily made part of the permanent symbol table.

These pseudo-ops can be used to eliminate unneeded symbols from the table, thus providing more storage for user symbols.

To append the following card reader IOT's to the symbol table, the programmer generates an ASCII tape of:

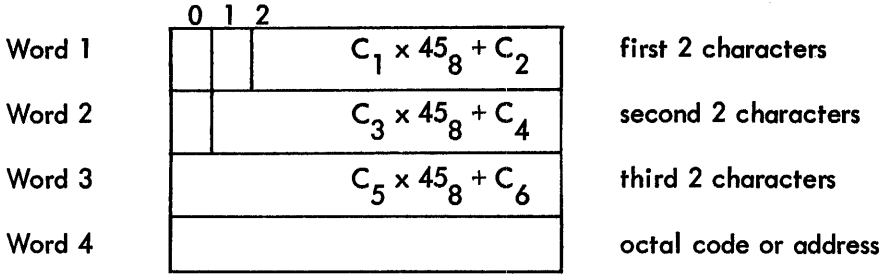
```
RCSF=6631
RCSP=6671
RCRD=6674
FIXTAB
PAUSE
```

The ASCII tape is then read into core ahead of the symbolic program tape during pass 1. The PAUSE pseudo-op stops assembly, and the Loader waits for the programmer to put the symbolic program tape into the tape reader and press CONTinue.

After each assembly, PAL-D's permanent symbol table is restored to contain only the basic symbols.

2.8.1 Internal Representation

Each permanent and user-defined symbol occupies four words (locations) in the symbol table storage area, as shown below.



where C_1, C_2, ..., C_6 represent the first character, second character, ..., sixth character respectively. (Symbols may consist of from one to six characters.) Bits 0 and 1 of word 1 and bit 0 of word 2 are system flags. With a permanent symbol, word 4 contains the octal code of the symbol; with a user-defined symbol, word 4 contains the address of the symbol. For example: the permanent symbol TAD is represented as follows.

Word 1 = $24_8 \times 45_8 + 01 = 1345_8$ or TA

Word 2 = $04_8 \times 45_8 + 00 = 224_8 + \underbrace{4000}_{\text{flag bit}} = 4224_8$ D

Word 3 = 0000

Word 4 = 1000 (octal code for TAD)

Note that the first digit of the USASCII octal code for each character is always trimmed by the assembler so that the character is represented using six bits of a word. For example, USASCII code for T is 324, it was trimmed to 24; A is 301, it was trimmed to 01; etc.

CHAPTER 3

PROGRAM PREPARATION AND ASSEMBLER OUTPUT

The source language tape (symbolic tape) is prepared using the Editor or an off-line ASR-33 Teletype.

3.1 PROGRAM TAPE

Since the Assembler ignores certain characters, these may be used freely to produce a more readable symbolic source tape. These useful characters are tab and form-feed.

The Assembler will also ignore extraneous spaces, carriage return-line feed combinations, rubouts, and blank tape.

The program body consists of statements and pseudo-ops. The program is terminated by the dollar sign ($). If the program is large, it may be segmented by use of the pseudo-op PAUSE. This often facilitates editing the source program since each section is physically smaller.

The Assembler initially sets the origin (current location counter) of the source program to 0200. The programmer may reset the current location counter by use of the asterisk.

The following two programs are identical except that format effectors were used in the second printout.

```
*200
/EXAMPLE OF FORMAT
/GENERATOR
BEGIN, 0/START OF PROGRAM
KCC
KSF/WAIT FOR FLAG
JMP .-1/FLAG NOT SET YET
KRB/READ IN CHARACTER
DCA CHAR
TAD CHAR
TAD MSPACE/IS IT A SPACE?
SNA CLA
HLT/YES
JMP BEGIN + 2/NO: INPUT AGAIN
CHAR, 0/TEMPORARY STORAGE
MSPACE, -240/-ASCII EQUIVALENT
/END OF EXAMPLE
$
```

```
*200
/EXAMPLE OF FORMAT
/GENERATOR
BEGIN,          0               /START OF PROGRAM
                KCC
                KSF             /WAIT FOR FLAG
                JMP .-1         /FLAG NOT SET YET
```

```
                KRB             /READ IN CHARACTER
                DCA CHAR
                TAD CHAR
                TAD MSPACE      /IS IT A SPACE?
                SNA CLA
                HLT             /YES
                JMP BEGIN+2     /NO: INPUT AGAIN
CHAR,           0               /TEMPORARY STORAGE
MSPACE,         -240            /-ASCII EQUIVALENT
/END OF EXAMPLE
$
```

Both of these programs will produce the same binary code. The second, however, is easier to read.

3.2 ASSEMBLY

PAL-D is a two-pass assembler with an optional third pass which produces a side-by-side assembly listing of the symbolic source statements, their octal equivalents, and assigned absolute addresses. When used with the TSS/8 time-sharing monitor the passes are invisible to the user. However, the user determines whether or not the third pass will be made by his response to PAL-D's OPTION: every (see Section 4.3.2).

3.2.1 Pass 1

During pass 1, PAL-D processes the source tape (or file) and places in its user's symbol table the definitions of all symbols used. The user's symbol table is printed (or punched) at the end of pass 2. If any symbols remain undefined at the end of pass 1, the US (Undefined Symbol) diagnostic is printed during pass 2 when the undefined symbol is encountered (see Error Diagnostics). The symbol table is printed (or punched) in alphabetical order on either the teleprinter or high-speed punch. The punched symbol table may be used to expand DDT-8s symbol table for use in program debugging. If the program listed above were assembled, PAL-D would output the following symbol table.

```
BEGIN   0200
CHAR    0213
MSPACE  0214
```

3.2.2 Pass 2

During pass 2, PAL-D processes the source tape (or file) and generates binary output using the symbol table equivalences defined during pass 1. The binary output may be loaded in core by the Disk Monitor System Binary Loader.

The binary coded tape (or file) consists of leader code, an origin setting, and data words. Every occurrence in the source program of an asterisk causes a new origin setting in the binary output. At the end of the binary coded tape, a binary checksum is produced and trailer code is generated.

When using the low speed paper tape punch, diagnostic messages are both typed and punched and will be preceded and followed by rubouts. The Binary Loader will ignore everything enclosed within rubouts.

3.2.3 Pass 3

During pass 3, PAL-D processes the source tape (or file) and prints out a side-by-side listing of the generated octal code and the original source language. If the program shown above were assembled, the pass 3 listing would be

```
                    *200
                    /EXAMPLE OF FORMAT
                    /GENERATOR
0200        0000    BEGIN,      0           /START OF PROGRAM
0201        6032                KCC
0202        6031                KSF         /WAIT FOR FLAG
0203        5202                JMP  .-1    /FLAG NOT SET YET
0204        6036                KRB         /READ IN CHARACTER
0205        3213                DCA CHAR
0206        1213                TAD CHAR
0207        1214                TAD MSPACE  /IS IT A SPACE?
0210        7650                SNA CLA
0211        7402                HLT         /YES
0212        5202                JMP BEGIN+2 /NO: INPUT AGAIN
0213        0000    CHAR,       0           /TEMPORARY STORAGE
0214        7540    MSPACE,     -240        /-ASCII EQUIVALENT
                    /END OF EXAMPLE
```

CHAPTER 4
LOADING AND ASSEMBLING PROCEDURES

The PAL-D Assembler is furnished on punched paper tape and is loaded and stored on the disk during system build time. Loading PAL-D in a TSS/8 system, is performed by the system manager and is described in detail in the TSS/8 System Manager's Guide, DEC-T8-MBZA-D. However, the user can at any time build a new system in a Disk Monitor system; therefore, complete loading procedures are detailed below.

4.1 DISK MONITOR SYSTEM

If the Disk Monitor is not present on your disk or DECtape, build it according to instructions in the Disk Monitor System manual, DEC-D8-SDAB-D.

4.1.1 Loading

The assembler is incorporated in the system by loading the paper tape into core using the disk Loader. Then the assembler may be saved on the disk or DECtape.

PAL-D is loaded into core in two passes as explained below. Disk system responses are underlined; non-underlined characters represent user-supplied data.

.LOAD ↵	call Loader from disk (↵ indicates carriage return)
*IN-R: ↵	input to be from high speed reader; T: would indicate input from Teletype reader
*	Loader found device R: valid
*OPT-2 ↵	two-pass load is specified
ST = ↵	control is to be returned to the Monitor after loading tape into core; 7600 ↵ would also transfer control to the Monitor after loading the tape

↑<↑P>↑<↑P>↑<↑P>↑<↑P>	Loader is waiting for user to put paper tape in reader and type ↑P.
	After reading tape into core, Loader waits for user to remove tape and type ↑P. If checksum error occurs, Monitor types ? in place of ↑.
	Loader is waiting for user to put paper tape in reader for second pass and type ↑P.
	After reading tape on second pass, Loader is waiting for user to remove tape and type ↑P. Again, checksum error will cause ? to be typed in place of ↑.

NOTE

↑P indicates CTRL-P, and < > indicates that the enclosed portion is not echoed (printed when the user types).

4.1.2 Saving

PAL-D may be saved on the system device as a system program. This is done by typing the following:

```
.SAVE PALD!0-7577;6200
      |    |    |
Program    Multiple   Entry
Name       Page Save  Point

{! System Program
{: User Program
```

The PAL-D Assembler is now saved as a system program on the system device. The programmer may now type PALD ↵ which brings the Assembler into core for use with symbolic source programs.

The user's core resident symbol table can hold 160_{10} user-defined symbols under the Disk Monitor System; 245_{10} under the TSS/8 Monitor System. This may be expanded by saving on the system device a user file named .SYM which can be used by PAL-D to store extra symbols. Each user-defined symbol occupies four words. The symbol table can be expanded by 128_{10} or 200_8 locations (one core page) by saving a file with the following statement.

.SAVE .SYM:0-177;0 ↵ (192 user symbols)

If a larger symbol table area is needed, simply specify additional pages, where each page saved provides storage for 32 additional symbols. For example:

.SAVE .SYM:0-377;0 ↵ (224 user symbols)

will save two core pages, and

.SAVE .SYM:0-1777;0 ↵ (416 user symbols)

will save eight core pages for symbol storage.

The preceding procedures are illustrated in Figure 1.

4.1.3 Assembling

PAL-D is transferred from the system device into core using the Monitor. One of the following methods is used depending upon the monitor type.

3.3.1 Disk Monitor System - The user begins by typing

.PALD ↵

PAL-D requests on output file by typing

*OUT-

The user selects the output device by typing

T: ↵ for the Teletype (low speed reader/punch), or

R: ↵ for the high speed reader/punch, or

S:name ↵ for output to the system device as file name

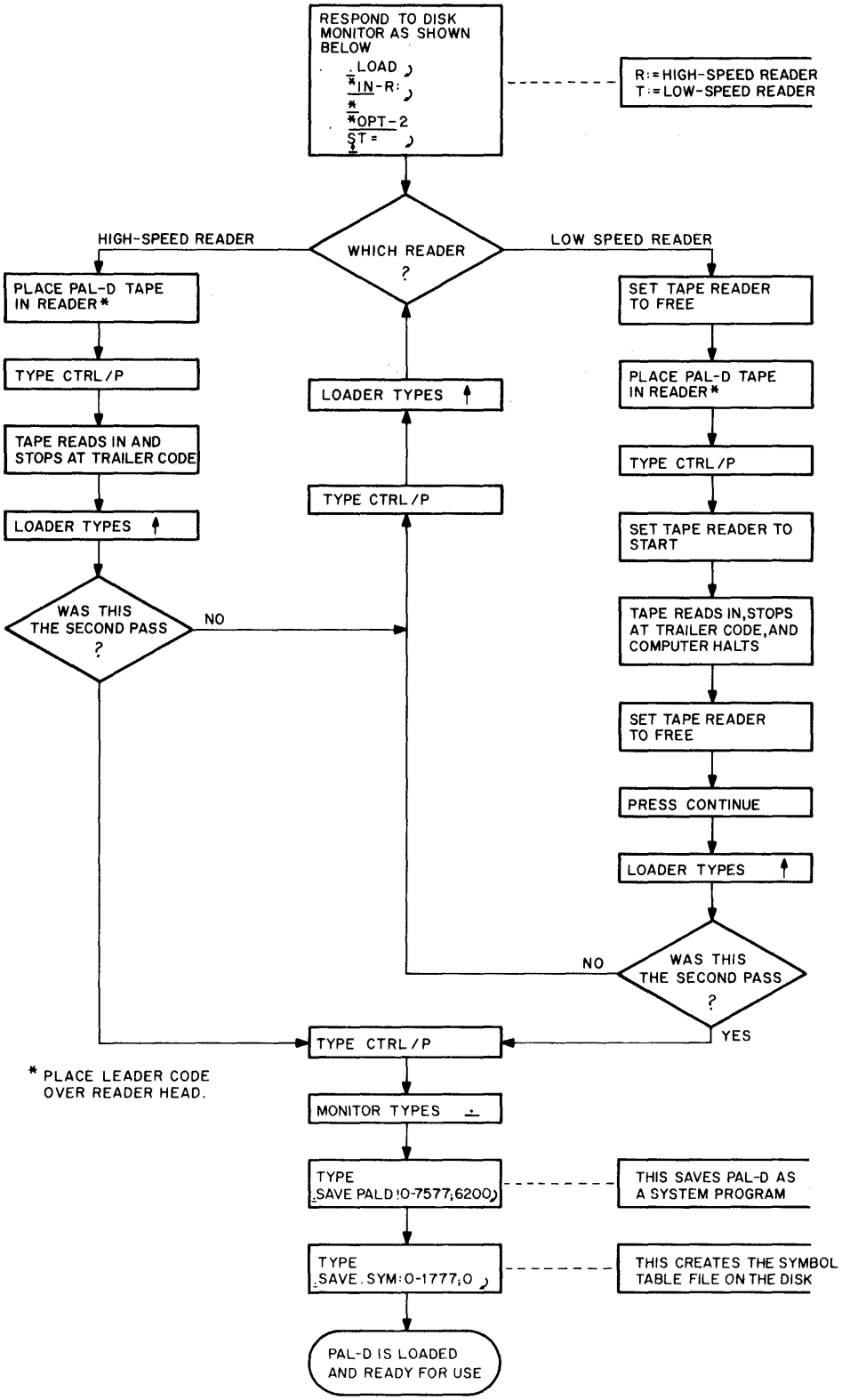


Figure 1 Loading and Saving PAL-D Using the Disk Monitor System

PAL-D now types

*IN-

and waits for the user to select the input files. Up to five input files may be specified (e.g., R:, R:, S:name, R:, R:↵), but in this example the user selected

R:↵ input from the high speed reader/punch

NOTE

PAL-D checks the validity of each selected file (i.e., valid only if the file was declared when building Monitor), and types * for each valid file and ? for an invalid file. When PAL-D finds an invalid file it returns control to the Monitor, in which case, the user must start again by calling PALD↵ .

When PAL-D is satisfied that the input file(s) is valid, it will request third pass listing option by typing

*OPT-

The user may type

T↵ meaning listing and symbols are to be produced on the Teletype, or

R↵ meaning listing and symbols are desired on high speed punch, or

↵ meaning symbols only (any other character means no third pass)

When the high speed punch is selected as a listing device, the alphabetic symbol table produced at the end of pass 2 is also produced on the high speed punch.

PAL-D will now proceed with the assembly, pausing only when user intervention is required (i.e., placing a new paper tape in the reader, turning on the punch, etc.). On these occasions, PAL-D will type an up-arrow (↑) on the Teletype to indicate user intervention is required. When the user has performed the necessary function and is ready to continue with the assembly, he types CTRL-P (which does not echo).

At the end of pass 2, PAL-D outputs the user's symbol table in alphabetical order (in addition to the assembled binary output). This symbol table listing may be terminated at any time by typing CTRL-P, and PAL-D will proceed to initiate pass 3, if requested.

Assembly may be terminated and control returned to the Monitor at any time by typing CTRL-C. When the assembly is complete, control will automatically be returned to the Monitor.

*With the low-speed reader: set reader to FREE, place tape in reader, type CTRL/P, and then set reader to START.

With the high-speed reader: place tape in reader and then type CTRL/P.

4.2 TSS/8 MONITOR SYSTEM

Assembling with PAL-D in TSS/8 requires no operator intervention between passes. The symbol table is typed out at the end of pass two and the listing at the end of pass three. The assembly may be terminated at any point by typing CTRL/C. Control will revert from PAL-D to the Monitor program which will type out a dot

.

and wait for the next instruction from the teletype. In the illustrations which follow, underlined characters are those typed out by the system; non-underlined characters represent user-supplied data. Time sharing assemblies are requested as follows.

In response to the monitor's dot

.

the user types the RUN (or simply R) command, a space and the name of the system program.

. R PALD ↵

PAL-D is brought into core and signals its readiness by requesting an input file name.

INPUT: BIN2 ↵

The user reply in this case was BIN2, a user symbol for a source program to be assembled.

PAL-D next requests the name of an output file.

OUTPUT: TYPE2 ↵

The user response was TYPE2, the name under which the assembled program will be stored.

Optionally, the user may type the RETURN key to specify no output file.

OUTPUT: ↵

This is useful in debugging. A program may be corrected and reassembled any number of times with production of an output file postponed until a satisfactory version is achieved.

PAL-D's final query is whether the user wants a program listing.

OPTION:

There are two effective responses only: N signifying No and ↵ (RETURN key) signifying Yes. When it receives the final response, PAL-D reads in the user source program from disk (source programs are stored prior to assembly) and proceeds with the assembly. After assembly, PAL-D returns control to the Monitor which types

.

and waits for the user to supply the next command.

NOTE

When running under the Disk Monitor system PAL-D requires a dollar sign ($) as the last entry in a source program. Under the TSS/8 Monitor PAL-D does not require one but if it does not find one it types a message to warn the user that his program may not be assembled properly by an assembly program other than time-sharing PAL-D.

The following listing was reproduced from a time sharing run. It illustrates the initial dialogue, the symbol table produced at the end of pass 2 (any error messages would also appear at this point) and the listing, in octal notation, produced at the end of pass 3.

INITIAL DIALOGUE

```
.R PALD

 INPUT:BIN2
OUTPUT:TYP2
OPTION:
```

SYMBOL TABLE

```
COUNT   0415
CRLF    0417
LOOP    0406
OUT     0425
REG     0416
START   0400
```

PROGRAM LISTING

```
                                /PROGRAM TO TYPE OUT "123456789"
                                *0400
0400   7200   START,  CLA
0401   4217           JMS CRLF
0402   1377           TAD (-12
0403   3215           DCA COUNT
0404   1376           TAD (260          /ASCII FOR ZERO
0405   3216           DCA REG
0406   1216   LOOP,   TAD REG
0407   4225           JMS OUT
0410   2216           ISZ REG
0411   2215           ISZ COUNT
0412   5206           JMP LOOP
0413   4217           JMS CRLF
0414   7402           HLT
0415   0000   COUNT,  0
0416   0000   REG,    0
0417   0000   CRLF,   0
0420   1375           TAD (215           /ASCII FOR CARRIAGE RETURN
0421   4225           JMS OUT
0422   1374           TAD (212           /ASCII FOR LINE FEED
0423   4225           JMS OUT
0424   5617           JMP I CRLF
0425   0000   OUT,    0
0426   6046           TLS
0427   6041           TSF
0430   5227           JMP .-1
0431   7200           CLA
0432   5625           JMP I OUT
```

LITERALS

```
0574   0212
0575   0215
0576   0260
0577   7766
↑BS
```

CHAPTER 5
ERROR DIAGNOSTICS

PAL-D makes many error checks as it processes source language statements. When an error is detected, the Assembler prints an error message. The format of the error messages is

ERROR CODE ADDRESS

where ERROR CODE is a two-letter code which specifies the type of error, and ADDRESS is either the absolute octal address where the error occurred or the address of the error relative to the last symbolic tag (if there was one) on the current page.

The programmer should examine each error indication to determine whether correction is required.

PAL-D's error messages are listed and explained below.

Error Code	Explanation
BE	Two PAL-D internal tables have overlapped - This situation can usually be corrected by decreasing the level of literal nesting or number of current page literals used prior to this point on the page.
DE	Systems device error - An error was detected when trying to read or write the system device; after three failures, control is returned to the Monitor.
DF	Systems device full - The capacity of the systems device has been exceeded; assembly is terminated and control is returned to the Monitor.
IC	Illegal character - An illegal character was encountered in other than a comment or TEXT field; the character is ignored and the assembly continued.
ID	Illegal redefinition of a symbol - An attempt was made to give a previously defined symbol a new value by other means than the equal sign; the symbol was not redefined.
IE	Illegal equals - An equal sign was used in the wrong context. Examples: TAD A +=B (the expression to the left of the equal sign is not a single symbol or, the expression to the right of the equal sign was not previously defined) A +B=C
II	Illegal indirect - An off-page reference was made; a link could not be generated because the indirect bit was already set.

Error Code	Explanation
	Example: *200 TAD I A ↵ . . . PAGE ↵ A, 7240 ↵
ND	The program terminator, $, is missing (with TSS/8 only).
PE	Current nonzero page exceeded - An attempt was made to a. override a literal with an instruction, or b. override an instruction with a literal; this can be corrected by (1) decreasing the number of literals on the page or (2) decreasing the number of instructions on the page.
PH	Phase error - PAL-D has received input files in an incorrect order; Assembly is terminated and control is returned to the Monitor.
SE	Symbol table exceeded - Assembly is terminated and control is returned to the Monitor; the symbol table may be expanded to contain up to 1184 user symbols by saving a file named .SYM on the system device.
US	Undefined symbol - A symbol has been processed during pass 2 that was not defined before the end of pass 1.
ZE	Page 0 exceeded - Same as PE except with reference to page 0.

APPENDIX A
USA SCII CHARACTER SET

Character	Code	Character	Code	Character	Code
A	301	0	260	!	241
B	302	1	261	"	242
C	303	2	262	#	243
D	304	3	263	$	244
E	305	4	264	%	245
F	306	5	265	&	246
G	307	6	266	'	247
H	310	7	267	(	250
I	311	8	270	)	251
J	312	9	271	*	252
K	313			+	253
L	314			,	254
M	315			-	255
N	316			.	256
O	317			/	257
P	320			:	272
Q	321			;	273
R	322			=	275
S	323			?	277
T	324			[	333
U	325			]	335
V	326			BELL	207
W	327			TAB	211
X	330			LINE FEED	212
Y	331			CARRIAGE-RETURN	215
Z	332			SPACE	240
				RUBOUT	377

APPENDIX B

SYMBOL LIST

Mnemonic	Code	Operation	Event Time
		MEMORY REFERENCE INSTRUCTIONS	
AND	0000	logical AND	
TAD	1000	2s complement add	
ISZ	2000	increment & skip if zero	
DCA	3000	deposit & clear AC	
JMS	4000	jump to subroutine	
JMP	5000	jump	
		GROUP 1 OPERATE MICROINSTRUCTIONS	
NOP	7000	no operation	1
IAC	7001	increment AC	3
RAL	7004	rotate AC & link left one	3
RTL	7006	rotate AC & link left two	3
RAR	7010	rotate AC & link right one	3
RTR	7012	rotate AC & link right two	3
CML	7020	complement link	2
CMA	7040	complement AC	2
CLL	7100	clear link	1
CLA	7200	clear AC	1
		GROUP 2 OPERATE MICROINSTRUCTIONS	
HLT	7402	halts the computer	4
OSR	7404	inclusive OR switch register with AC	3
SKP	7410	skip unconditionally	1
SNL	7420	skip on nonzero link	1
SZL	7430	skip on zero link	1
SZA	7440	skip on zero AC	1
SNA	7450	skip on nonzero AC	1
SMA	7500	skip on minus AC	1
SPA	7510	skip on plus AC (zero is positive)	1
		COMBINED OPERATE MICROINSTRUCTIONS	
CIA	7041	complement & increment AC	1
STL	7120	set link to 1	1
GLK	7204	get link (put link in AC, bit 11)	1
STA	7240	set AC = -1	1
LAS	7604	load AC with switch register	1
		PSEUDO-OPERATORS	
DECIMAL			
EXPUNGE			
FIELD			
FIXTAB			
I			
OCTAL			
PAGE			

PSEUDO-OPERATORS

PAUSE
TEXT
XLIST
Z

Mnemonic	Code	Operation	Event Time
IOT MICROINSTRUCTIONS FOR DISK MONITOR			
Program Interrupt			
ION	6001	turn interrupt on	
IOF	6002	turn interrupt off	
Keyboard/Reader			
KSF	6031	skip if keyboard/reader flag = 1	
KCC	6032	clear AC & keyboard/reader flag	
KRS	6034	read keyboard/reader buffer	
KRB	6036	clear AC & read keyboard buffer, & clear keyboard flag	
Teleprinter/Punch			
TSF	6041	skip if teleprinter/punch flag = 1	
TCF	6042	clear teleprinter/punch flag	
TPC	6044	load teleprinter/punch buffer, select & print	
TLS	6046	load teleprinter/punch buffer, select & print, and clear teleprinter/punch flag	
High-Speed Reader (Type PC02)			
RSF	6011	skip if reader flag = 1	
RRB	6012	read reader buffer & clear flag	
RFC	6014	clear flag & buffer & fetch character	
High-Speed Punch (Type PC03)			
PSF	6021	skip if punch flag = 1	
PCF	6022	clear flag & buffer	
PPC	6024	load buffer & punch character	
PLS	6026	clear flag & buffer, load & punch	
Disk File and Control (type DF32)			
DCMA	6601	clear disk memory request & interrupt flags	
DMAR	6603	load disk from AC, clear AC, read into core, clear interrupt flag	
DMAW	6605	load disk from AC, write onto disk from core, clear interrupt flag	
DCEA	6611	clear disk extended address & memory address extension register	
DSAC	6612	skip if address confirmed flag = 1	
DEAL	6615	clear disk extended address & memory address extension register & load same from AC	
DEAC	6616	clear AC, load AC from disk extended address register, skip if address confirmed flag = 1	
DFSE	6621	skip if parity error, data request late, or write lock switch flag = 0 (no error)	

Mnemonic	Code	Operation	Event Time
DFSC	6622	skip if completion flag = 1 (date transfer completed)	
DMAC	6626	clear AC, load AC from disk memory address register	
DECtape Transport (Type TU55) and Control (Type TC01)			
DTRA	6761	read status register A	1
DTCA	6762	clear status register A	2
DTXA	6764	load status register A	3
DTSF	6771	skip on flags	1
DTRB	6772	read status register B	2
DTLB	6774	load status register B	3
Memory Extension Control (Type 183)			
CDF	62n1	change to data field n	1
CIF	62n2	change to instruction field n	1
RDF	6214	read data field into AC 6-8	1
RIF	6224	read instruction field into AC 6-8	1
RMF	6244	restore memory field	1
RIB	6234	read interrupt buffer	1

IOT MICROINSTRUCTIONS FOR TSS/8 MONITOR

Program Interrupt			
IOT	6000	(See Time-Sharing System User's Guide, DEC-T8-MRFB-D.)	
Keyboard/Reader			
KSF	6031	skip if keyboard/reader flag = 1	
KCC	6032	clear AC & keyboard/reader flag	
KRS	6034	read keyboard/reader buffer	
KRB	6036	clear AC & read keyboard buffer, & clear keyboard flag	
KSB	6400	set keyboard break	
SBC	6401	set buffer control flags	
KSR	6030	read keyboard string	
Teleprinter/Punch			
TSF	6041	skip if teleprinter/punch flag = 1	
TCF	6042	clear teleprinter/punch flag	
TPC	6044	load teleprinter/punch buffer, select & print	
TLS	6046	load teleprinter/punch buffer, select & print, and clear teleprinter/punch flag	
SAS	6040	send a string	
High-Speed Reader (Type PC02)			
RSF	6011	skip if reader flag = 1	
RRB	6012	read reader buffer & clear flag	
RFC	6014	clear flag & buffer & fetch character	
RRS	6010	read reader string	

Mnemonic	Code	Operation	Event Time
High-Speed Punch (Type PC03)			
PSF	6021	skip if punch flag = 1	
PCF	6022	clear flag & buffer	
PPC	6024	load buffer & punch character	
PLS	6026	clear flag & buffer, load & punch	
PST	6020	punch string	
DECtape Transport (Type TU55) and Control (Type TC01)			
DTXA	6764	load status register A	3
DTSF	6771	skip on flags	1
DTRB	6772	read status register B	2
Program Control			
URT	6411	user run time	
TOD	6412	time of day	
RCR	6413	return clock rate	
DATE	6414	Date	
SYN	6415	quantum synchronization	
STM	6416	set timer	
TSS	6420	skip on TSS/8	
USE	6421	user	
SSW	6430	set switch register	
CKS	6200	check status	
ASD	6440	assign device	
REL	6442	release device	
DUP	6402	duplex	
CON	6422	console	
File Control			
REN	6600	Rename File	
OPEN	6601	Open File	
CLOS	6602	Close File	
RFILE	6603	Read File	
PROT	6604	Protect File	
WFILE	6605	Write File	
CRF	6610	Create File	
EXT	6611	Extend File	
RED	6612	Reduce File	
FINF	6613	File Information	
SIZE	6614	Segment Size	
SEGS	6406	Segment Count	
ACT	6617	Account Number	
WHO	6616	Who	

PAL-D INDEX

PAL-D ASSEMBLER
DEC-D8-ASAB-D

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.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

_____

_____

_____

_____

_____

Did you find errors in this manual? _____

_____

_____

_____

_____

How can this manual be improved? _____

_____

_____

_____

_____

_____

DEC also strives to keep its customers informed of current DEC software and publications. Thus, the following periodically distributed publications are available upon request. Please check the appropriate boxes for a current issue of the publication(s) desired.

☐ Software Manual Update, a quarterly collection of revisions to current software manuals.

☐ User's Bookshelf, a bibliography of current software manuals.

☐ Program Library Price List, a list of currently available software programs and manuals.

Please describe your position. _____

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754