PAL III SYMBOLIC ASSEMBLER PROGRAMMING MANUAL

PDP-8

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS
PREFACE

The PDP-8 comes to the user complete with an extensive selection of system programs and routines making the full data processing capability of the new computer immediately available to each user, eliminating many commonly experienced initial programming delays.

The programs described in these abstracts come from two sources, past programming effort on the PDP-5 computer, and present and continuing programming effort on the PDP-8. Thus the PDP-8 programming system takes advantage of the many man-years of program development and field testing by PDP-5 users.

Although in many cases PDP-8 programs originated as PDP-5 programs, all utility and functional program documentation is issued in a new, recursive format introduced with the PDP-8.

Programs written by users of either the PDP-5 or the PDP-8 and submitted to the users' library (DECUS - Digital Equipment Corporation Users' Society) are immediately available to PDP-8 users.

Consequently, users of either computer can take immediate advantage of the continuing program developments for the other.
## CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2</td>
<td>ILLUSTRATIONS OF PDP-8 ASSEMBLER FEATURES</td>
<td>2-1</td>
</tr>
<tr>
<td></td>
<td>The Location Counter</td>
<td>2-1</td>
</tr>
<tr>
<td></td>
<td>Coding Illustrations</td>
<td>2-1</td>
</tr>
<tr>
<td>3</td>
<td>THE SOURCE LANGUAGE</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>The Character Set</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Letters</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Digits</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Punctuation Characters</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Ignored Characters</td>
<td>3-2</td>
</tr>
<tr>
<td></td>
<td>Illegal Characters</td>
<td>3-2</td>
</tr>
<tr>
<td></td>
<td>Elements</td>
<td>3-2</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>3-2</td>
</tr>
<tr>
<td></td>
<td>Symbol</td>
<td>3-3</td>
</tr>
<tr>
<td></td>
<td>Parameter Assignments</td>
<td>3-3</td>
</tr>
<tr>
<td></td>
<td>Symbol Definition</td>
<td>3-4</td>
</tr>
<tr>
<td></td>
<td>Expressions</td>
<td>3-5</td>
</tr>
<tr>
<td></td>
<td>Current Address Indicator</td>
<td>3-8</td>
</tr>
<tr>
<td></td>
<td>Comments</td>
<td>3-9</td>
</tr>
<tr>
<td></td>
<td>Pseudo-Instructions</td>
<td>3-9</td>
</tr>
<tr>
<td>4</td>
<td>PROGRAM PREPARATION AND ASSEMBLER OUTPUT</td>
<td>4-1</td>
</tr>
<tr>
<td></td>
<td>Program Tape</td>
<td>4-1</td>
</tr>
<tr>
<td>5</td>
<td>OPERATING INSTRUCTIONS</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>5-2</td>
</tr>
<tr>
<td>6</td>
<td>SYMBOL TABLE ALTERATION</td>
<td>6-1</td>
</tr>
<tr>
<td>Appendix</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>SYMBOL LISTS</td>
<td>A1-1</td>
</tr>
<tr>
<td>2</td>
<td>ASCII CHARACTER SET</td>
<td>A2-1</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

The use of an assembly program has become standard practice in the programming of digital computers. Use of an assembler permits a programmer to code in a more convenient language than basic machine code. The advantages of this practice are widely recognized: Easily recognized mnemonic codes are used instead of numeric codes; instructions or data may be referred to by a symbolic name; decimal data may be used as such with the assembler making the required decimal-to-binary conversion; programs may be altered without extensive changes in the source language; and debugging is simplified.

The basic process performed by the Assembler is the substitution of numeric values for symbols, according to associations found in the symbol table. In addition, the user may request that the Assembler itself assign values to the user's own symbols at assembly time. These symbols are normally used to name memory locations, which may then be referenced by name.

The ability to use mnemonic names to represent machine instructions is of great value. The name TAD reminds the user of the Two's complement ADdition instruction, while the number 1000 does not. Consequently, the instructions are easier to remember when mnemonics are used. The same is true of location names. It is much easier to associate the name TOTAL with the location containing the accumulated total than it is to remember that location 1374 contains the total.

Another advantage is that, since the assignment of absolute numbers to symbolic locations is done by the Assembler, the updating of a program by adding or removing instructions is simplified.

In addition to translating statements directly into their binary equivalents, the Assembler will accept instructions for performing translations. These instructions may not look different from other instructions, but they do not generate binary codes. For this reason, they are referred to as pseudo-instructions. For example, the pseudo-instruction DECIMAL tells the Assembler that all numbers following in the program are to be taken as decimal rather than as octal. This instruction is important to the assembly process but has no binary equivalent in the object program. Certain other features of assembly can be directed to the Assembler by the setting of the switch register, abbreviated SR.

The PDP-8 Assembly system consists of the Assembler (PAL III) and the Binary Loader (Digital-8-2-U). A source program prepared in the source language using ASCII code is translated by the Assembler into a binary object tape in two passes through the Assembler. The object binary tape is loaded by the Binary Loader into the computer ready for execution.
During the first pass of the assembly, all symbols are defined and placed in the Assembler's symbol table. During the second pass, the binary equivalents of the input source language are generated and punched. The Assembler has an optional third pass, which produces an "assembly listing," or a listing with the location, generated binary, and source code side by side on a line.

The PDP-8 Assembly system also includes the Symbolic Tape Editor (Digital-8-1-S) for altering or editing the source language tape; the DEC Debugging Tape (DDT-8, Digital-8-4-S) for debugging the object program by communicating with it in the source language, and various other utility programs such as dumps, etc.

The Assembler requires a basic PDP-8 system consisting of the 33 ASR Tape Reader and Punch and a 4K core memory. The Assembler can use either the 750C Photo-Electric Reader, the 75E High-Speed Punch, or both. The basic Assembler allows 590 user symbols when using the 33 ASR and allows 495 user symbols when using the photoelectric reader. The Extended Assembler contains additional symbols for all optional devices. This symbol list is to be found in the Appendix.
CHAPTER 2

ILLUSTRATIONS OF PDP-8 ASSEMBLER FEATURES

THE LOCATION COUNTER

In general, statements generate 12-bit binary words which are placed into consecutive memory locations when the object tape is loaded. The location counter is a register used by the PDP-8 Assembler to keep track of the next memory location available. It is updated after processing each statement. The location counter may be explicitly set by an element or expression preceded by an asterisk. The element or expression following the asterisk sets the current location counter to the value of that element or expression. Subsequent instructions are assembled into subsequent locations.

Example:

*300

The next instruction would be placed in location 300. The location counter is initially set to 0200.

CODING ILLUSTRATIONS

To illustrate some of the features of the PDP-8 Assembler, a small routine has been chosen and coded in a number of different ways. The routine continually adds 1 to the contents of a location until the result is positive, then halts. The instructions used are represented as their octal codes (more compact than the binary actually used). The number being incremented is in location 170. The notation C(A) means the contents of location A.

*100  1170 /C(170) INTO AC
*101  7001 /ADD 1 TO AC
*102  3170 /STORE IN LOCATION 170
*103  1170 /FETCH C(170)
*104  7710 /SKIP ON POSITIVE AC, CLEAR AC
*105  5100 /JUMP TO LOCATION 100
*106  7402 /HALT
*170  0 /WILL CONTAIN NUMBER TO BE INCREMENTED

Since the location counter is automatically incremented, specifying sequential addresses could have been avoided after the first address in the progression. In addition, the names of the PDP-8 instructions could have been used in place of the octal codes. The octal representation of these instructions is substituted by the Assembler whenever symbols appear in the program.
Example 2:

*100

<table>
<thead>
<tr>
<th>TAD</th>
<th>170</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAC</td>
<td></td>
</tr>
<tr>
<td>DCA</td>
<td>170</td>
</tr>
<tr>
<td>TAD</td>
<td>170</td>
</tr>
<tr>
<td>SPA</td>
<td>CLA</td>
</tr>
<tr>
<td>JMP</td>
<td>100</td>
</tr>
<tr>
<td>HLT</td>
<td></td>
</tr>
</tbody>
</table>

*170

∅

The same program could have been written using symbolic address tags. The comma after the symbol A indicates to the Assembler that the location in which it places the instruction TAD B is to be named A. Information associating the symbol A with the number of actual locations is placed in the Assembler's symbol table. Consequently, when processing the instruction JMP A, the Assembler finds the symbols JMP and A in the symbol table and uses these values to form the binary equivalent of the instruction JMP A.

Example 3:

*100

A,

<table>
<thead>
<tr>
<th>TAD B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IAC</td>
<td></td>
</tr>
<tr>
<td>DCA B</td>
<td></td>
</tr>
<tr>
<td>TAD B</td>
<td></td>
</tr>
<tr>
<td>SPA CLA</td>
<td></td>
</tr>
<tr>
<td>JMP A</td>
<td></td>
</tr>
<tr>
<td>HLT</td>
<td></td>
</tr>
</tbody>
</table>

*170

∅

B,

Unless the user specifically wanted to use location 170 for storage, he could let the Assembler assign the location.

Example 4:

*100

A,

<table>
<thead>
<tr>
<th>TAD B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IAC</td>
<td></td>
</tr>
<tr>
<td>DCA B</td>
<td></td>
</tr>
<tr>
<td>TAD B</td>
<td></td>
</tr>
<tr>
<td>SPA CLA</td>
<td></td>
</tr>
<tr>
<td>JMP A</td>
<td></td>
</tr>
<tr>
<td>HLT</td>
<td></td>
</tr>
</tbody>
</table>

B,

∅
CHAPTER 3

THE SOURCE LANGUAGE

This chapter explains the features of the ASCII source language available to the user of PAL III.

THE CHARACTER SET

Letters
A B C D E...X Y Z

Digits
1 2 3 4 5 6 7 8 9 Ø

Punctuation Characters

Since a number of characters are invisible (i.e. nonprinting), the following notation is used to represent them in the examples:

\[ \begin{align*}
\rightarrow & \quad \text{space} \\
\rightarrow & \quad \text{tab} \\
\rightarrow & \quad \text{carriage return}
\end{align*} \]

The following characters are used to specify operations to be performed upon symbols or numbers:

<table>
<thead>
<tr>
<th>Character</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>\rightarrow</td>
<td>space</td>
</tr>
<tr>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>-</td>
<td>minus</td>
</tr>
<tr>
<td>\rightarrow</td>
<td>carriage return</td>
</tr>
<tr>
<td>\rightarrow</td>
<td>tab</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>=</td>
<td>equals</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>$</td>
<td>dollar sign</td>
</tr>
</tbody>
</table>
. point has value equal to current location counter
/ slash indicates start of a comment

Ignored Characters

form feed end of a logical page of a source program (See Symbolic Editor 8-1-5)
blank tape used for leader/trailer
rubouts used for deleting characters
code 280 used for leader/trailer
line feed follows carriage return

Illegal Characters

All other characters are illegal and cause the Illegal Character error printout: IC dddd AT dddd during PASS1. The first number is the value of the offending character, and the second is the value of the current location counter where it occurred. Illegal characters are ignored.

ELEMENTS

Any group of letters, digits, and punctuation which represents binary values less than $2^{12}$ is an element.

Number

Any sequence of numbers delimited by punctuation characters forms a number.

Example:

1
12
4372

The radix control pseudo-instructions indicate to the Assembler the radix to be used in number interpretation. The pseudo-instruction DECIMAL indicates that all numbers are to be interpreted as decimal until the next occurrence of the pseudo-instruction OCTAL.

The pseudo-instruction OCTAL indicates that all numbers are to be interpreted as octal until the next occurrence of the pseudo-instruction DECIMAL. The radix is initially set to octal and remains octal unless otherwise specified.
Symbol

Any sequence of letters and digits beginning with a letter and delimited by punctuation characters is a symbol. Although a symbol may be any length, only the first six characters are considered, and any additional characters are ignored; symbols which are identical in their first six characters are considered identical.

The Assembler has in its permanent symbol table definitions of the symbols for all PDP-8 operation codes, operate commands, and many IOT commands (see the Appendix for a complete list). These may be used without prior definition by the user.

Examples:

JMS is a symbol whose value of 40000 is taken from the operation code definitions.

A is a user-created symbol. When used as a symbolic address tag, its value is the address of the instruction it tags. This value is assigned by the Assembler.

PARAMETER ASSIGNMENTS

A parameter may be assigned by use of the equal sign. The symbol to the left of the equal sign is assigned the value of the expression on the right.

Examples:

A=6
EXIT=RETURN=JMP 1 0

Symbols defined by use of the equal sign may be used in any valid expression.

Example:

A=100
B=400
A+B has the value 500
TAD A has the value 1100

If the expression to the left of the equal sign has already been defined, the ReDefinition diagnostic:

RD XXXXXX AT dddd

Will be typed where XXXXXX is the symbol's name and dddd is the contents of the current location counter at the point of redefinition. The new value will be stored in the symbol table.
Example:

\[ *100 \]
CLA=7600

will cause the diagnostic:

RD CLA AT 0100

Whenever CLA is used after this point, it will have the value 7600.

**SYMBOL DEFINITION**

A symbol may be defined by the user in one of two ways

(1) by use of parameter assignment

Example:

DISMIS=JMP I 6

and (2) by use of the comma

When a symbol is terminated by a comma, it is assigned a value equal to the current location counter. If it is defined more than once in this manner, the Assembler will type the duplicate tag diagnostic:

DT XXXXXX AT dddd

where XXXXXX is the symbol, and dddd is the current location counter at the second occurrence of the attempted symbol definition. The symbol is not redefined.

Example:

\[ *300 \]
START, TAD A
       DCA COUNTER
CONTIN, JMS LEAVE
          JMP START
A, -74
COUNTER, 6
START, CLA CLL
\[

The symbol "START" would have a value of 0300, the symbol "CONTIN" would have a value of 0302, the symbol "A" would have a value of 0304, the symbol "COUNTER" (considered by the Assembler to be COUNT) would have a value of 0305, and when the Assembler processed the next line, it would type during PASS1:

3-4
DT START AT $306

Since the first PASS of PAL III is used to define all symbols in the symbol table, the Assembler will type a diagnostic if, at the end of PASS1, there are any symbols remaining undefined. For example:

```
*7170
A,   TAD C
    CLA CMA
    HLT
    JMP A1
C,    0
$
```

would produce the Undefined Address diagnostic:

```
UA XXXXXX AT dddd
```

where XXXXXX is the symbol and dddd is the location at which it was first seen. The entire symbol table is printed at the end of PASS1. In the case of the above example, this would be:

```
A  7170
UA A1 AT 7173
C  7174
```

If, during PASS1, PAL III detects that its symbol table is full (in other words, that there is no more memory space to store symbols and their associated values), the Symbol Table full diagnostic:

```
ST XXXXXX AT dddd
```

is typed. XXXXXX is the symbol that caused overflow, and dddd is the current location when the overflow occurred. The Assembler halts and may not be restarted. The source program should be segmented, or more address arithmetic used, to reduce the number of symbols. PAL III's symbol capacity is:

Using 33 ASR; 655 symbols. The basic symbol table contains 65 symbols (see Appendix) leaving 590 user-defined symbols. Using the 750 Photo-Electric Reader; 560 symbols. The basic symbol table contains 65 symbols leaving 495 user-defined symbols.

**EXPRESSIONS**

Symbols and numbers are combined with certain operators to form expressions. There are three operators:

```
+     plus       this signifies 2's complement addition
-     minus      this signifies 2's complement subtraction
\[   space       space is interpreted in context. Since a PDP-8 instruction has an operation code of three bits as well as an indirect bit, a page bit, and seven address bits, the Assembler must combine memory reference instructions
```
in a manner somewhat different from the way in which it combines operate or IOET instructions. The Assembler accomplishes this by differentiating the symbols in its permanent symbol table. The following symbols are used as memory reference instruction op codes:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>0000</td>
<td>logical AND</td>
</tr>
<tr>
<td>TAD</td>
<td>1000</td>
<td>Two's complement ADdition</td>
</tr>
<tr>
<td>ISZ</td>
<td>2000</td>
<td>Index and Skip if Zero</td>
</tr>
<tr>
<td>DCA</td>
<td>3000</td>
<td>Deposit and Clear Accumulator</td>
</tr>
<tr>
<td>JMS</td>
<td>4000</td>
<td>Jump to Subroutine</td>
</tr>
<tr>
<td>JMP</td>
<td>5000</td>
<td>Jump</td>
</tr>
<tr>
<td>FADD</td>
<td>1000</td>
<td>Floating ADDition</td>
</tr>
<tr>
<td>FSUB</td>
<td>2000</td>
<td>Floating SUBtraction</td>
</tr>
<tr>
<td>FMPY</td>
<td>3000</td>
<td>Floating MUltiPY</td>
</tr>
<tr>
<td>FDIV</td>
<td>4000</td>
<td>Floating DIVide</td>
</tr>
<tr>
<td>FGET</td>
<td>5000</td>
<td>Floating GET</td>
</tr>
<tr>
<td>FPUT</td>
<td>6000</td>
<td>Floating PUT</td>
</tr>
<tr>
<td>FNOR</td>
<td>7000</td>
<td>Floating NORMalize</td>
</tr>
<tr>
<td>FEXT</td>
<td>8000</td>
<td>Floating EXIT</td>
</tr>
</tbody>
</table>

When the Assembler has processed one of these symbols, the space acts as an address field delimiter:

```
*4100
A,  CLA
```

A has the value 4101, JMP has the value 5000, and the space acts as a field delimiter. These symbols are combined as follows:

```
A   100 001 000 001
JMP 101 000 000 000
```

The seven address bits of A are taken, i.e.:

```
000 001 000 001
```

The remaining bits of the address are tested to see if they are zero's (page zero reference); if they are not, the current page bit is set:

```
000 011 000 001
```

The operation code is then ORed into the expression to form:

```
101 011 000 001
```

or, written more concisely:

```
5301
```
In addition to the above outlined tests, the page bits of the address field are compared with the page bits of the current location counter. If the page bits of the address field are nonzero and do not equal the page bits of the current location counter, an out-of-page reference is being attempted and the Illegal Reference diagnostic is printed on PASS2 or PASS3.

For example:

```
*4100
A,  CLA CLL
: *
*7200
JMP A
```

The symbol in the address field of the jump instruction has a value of 4100 while the current location counter, i.e., the address where the instruction will be placed in memory, has a value of 7200. This instruction is illegal on the PDP-8 and will be flagged during PASS2 or PASS3 by the Illegal Reference diagnostic:

```
IR 4100 AT 7200
```

The value 5300 would be assembled at location 7200.

The symbol I caused the indirect bit (bit 3) to be set in a memory reference instruction: For example:

```
DCA I 10
```

would produce:

```
011 100 001 000
```

or:

```
3410
```

When a space occurs in an expression that does not contain a memory reference instruction op code, it means inclusive OR:

For example:

```
CLA CLL
```

the symbol CLA has a value of 7200 and the symbol CLL has a value of 7100; CLA CLL would produce 7300.

User-defined symbols are treated as nonmemory reference instructions (see Pseudo-Instructions).
For example:

\[
\begin{align*}
A &= 333 \\
*222 & \quad B, \text{ CLA}
\end{align*}
\]

Then the expressions and their values are shown below:

\[
\begin{align*}
A+B & \quad 0555 \\
A-B & \quad 0111 \\
A-B & \quad 0333 \\
-A & \quad 7445 \\
1-B & \quad 7557 \\
B-1 & \quad 0221 \\
-A & \quad 7707 \\
\text{etc.}
\end{align*}
\]

An expression is terminated by either a carriage-return (\text{\&}) or a semicolon (;). If any information was generated to be loaded, the current location counter is incremented.

Example:

\[
\begin{align*}
\text{RAR; RTR; CMA}\text{\&}
\end{align*}
\]

Produces three registers of information and the current location counter is incremented after each expression. The statement:

\[
\text{HALT=HLT CLA}\text{\&}
\]

produces no information to be loaded (it produces an association in the Assembler's symbol table) and hence does not increment the current location counter.

\[
\begin{align*}
*4721 \\
\text{TEMP, } & \quad \text{ } \\
\text{TEM2, } & \quad \overline{0}
\end{align*}
\]

The current location counter is not incremented after the line \text{TEMP, } and hence the two symbols \text{TEMP} and \text{TEM2} are assigned the same value, in this case 4721.

\textbf{CURRENT ADDRESS INDICATOR}

The single character period (\text{.}) has, at all times, a value equal to the value of the current location counter.

It may be used as any number or symbol (except to the left of the equal sign).

Example:

\[
\begin{align*}
*2000 \\
\text{JMP } & \quad .+2
\end{align*}
\]
is equivalent to JMP 202.

\[
\begin{align*}
&*300 \quad +2400 \\
&\text{would produce, in register 300, the quantity 2700}
\end{align*}
\]

Example:

\[
\begin{align*}
&*2200 \\
&\text{CALL}=\text{JMS 1 .} \\
&27
\end{align*}
\]

Since the second line, CALL=JMS 1 , does not increment the current location counter, 0027 would be placed in register 2200 and CALL would have the value of 100 110 0002 or 46008.

The properties of the character (.) have been slightly changed; so that, it now acts as a terminator. Previously, PAL III would neither diagnose nor correctly assemble expressions such as: JMP. (where there is no space between the P and the .) PAL III now treats this (JMP.) as if it were this (JMP .)

\section*{COMMENTS}

A comment field is indicated by the slash (/) character. The Assembler will ignore everything from the slash to the next carriage return.

Example:

\[
\text{CLA} \quad /\text{THIS IS A COMMENT}
\]

\section*{PSEUDO-INSTRUCTIONS}

There are several pseudo-instructions that are used to direct the Assembler. These are:

\begin{itemize}
\item **DECIMAL** Set the current radix to decimal
\item **OCTAL** Set the current radix to octal
\item **PAUSE** Stop the Assembler. The current pass is not terminated. PAUSE must be at the physical end of the program tape segment as the reader routines are buffered and the buffer is emptied when PAUSE is detected. The assembly is continued by depressing CONTINUE.
\item **FIELD EXPRESSION** Causes a field setting to be punched during PASS2. This is recognized by the Extended Memory Loader (Digital-B-2A-U) and causes all subsequent information to be loaded into the field specified by the expression. The expression must be between 0 and 7, inclusive.
\item **EXPUNGE** Expunge the entire symbol table except for the pseudo-instructions.
\item **FIXTAB** Fix the current symbol table. Symbols that have been fixed are not printed in the symbol table at the end of PASS1 or PASS 3.
\end{itemize}

3-9
FIXMRI  Fix memory reference instruction. This may be given only after
EXPUNGE. It tells the Assembler that the following symbol definition
is a memory reference instruction and is to be treated as described under
Expressions.

Example:

EXPUNGE
FIXMRI TAD=1000
FIXMRI DCA=30000
CLA=72000
FIXTAB
PAUSE

When this program segment is read into the Assembler during PASS1, all symbol definitions are
deleted and the three symbols listed are added to the table.

This process is often performed to alter the Assembler's symbol table so that it contains only those
symbols that will be used. This may increase the Assembler's capacity for other user-defined symbols.
CHAPTER 4

PROGRAM PREPARATION AND ASSEMBLER OUTPUT

The source language tape (symbolic tape) is prepared in ASCII code on 8-channel punched paper tape using an off-line Teletype or the on-line Symbolic Tape Editor (Digital-8-1-5). In general, a program should begin with leader code which may be blank tape, code 200, or rubouts.

PROGRAM TAPE

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

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

The program body consists of statements and pseudo-instructions. The program is terminated by the dollar sign followed by some trailer code. If the program is large, it may be segmented by use of the pseudo-instruction PAUSE. This often facilitates the editing of the source program since each section will be physically smaller.

The Assembler initially sets its current location counter to 200. This is reset whenever the asterisk is processed.

During PASS 1, all illegal characters cause a diagnostic to be printed. The character is ignored.

The following two programs are identical:

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

4-1
Both these programs are identical and produce the same binary code. The second, however, is easier to read.

During PASS1, the Assembler reads the source tape and defines all symbols used. The user's symbol table is printed (or punched) at the end of PASS1. If any symbols remain undefined, the UA diagnostic is printed. The symbol table is printed in alphabetic order. If the program listed above were assembled, the PASS1 output would be:

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

During PASS2, the Assembler reads the source tape and generates the binary code using the symbol table equivalences defined during PASS1. The binary tape that is punched may be loaded by the Binary Loader (Digital-8-2-U). This binary tape consists of leader code, an origin setting, and then data words. Every occurrence of an asterisk expression causes a new origin to be punched on the tape and resets the Assembler's current location counter. At the end of PASS2, the checksum is punched on the binary tape and trailer code is generated. During PASS2, the Assembler may diagnose an Illegal Reference. When using the 33 ASR Punch, the diagnostic will be both typed and punched and will be preceded and followed by rubouts. The Binary Loader will ignore everything that has been punched on a tape between rubouts.

During PASS3, the Assembler reads the source tape and generates the code from the source statements. The assembly listing is typed (or punched). It consists of the current location counter, the generated code in octal, and the source statement. The symbol table is typed at the end of the pass. If the program listed above were assembled, the PASS3 output would be:
*200

/EXAMPLE OF FORMAT
/GENERATOR

| 0200 | 0000 | BEGIN, ø | /START OF PROGRAM |
| 0201 | 6032 | KCC      |
| 0202 | 6031 | KSF      | /WAIT FOR FLAG |
| 0203 | 5202 | JMP.-1   | /FLAG NOT SET YET |
| 0204 | 6036 | KR      | /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, ø  | /TEMPORARY STORAGE |
| 0214 | 7540 | MSPACE,-240 | /ASCII EQUIVALENT |

BEGIN 0200
CHAR 0213
MSPACE 0214
CHAPTER 5

OPERATING INSTRUCTIONS

The PAL III Assembler is provided as a binary tape. This is loaded into the PDP-8 memory by means of the Binary Loader, using either the 33 ASR Reader or the 750C Photo-Electric Reader (see Digital-8-2-U). The Assembler will use either the 33 ASR Reader or the photo-electric reader to read the source language tape, and it will use either the 33 ASR Punch or the 75E Punch for output. The selection of I/O devices is made by the Assembler when it is started. The source language tape must be in the proper reader, with the reader and punch turned on. When using the high-speed punch, the symbol table will be typed on the 33 ASR if bit 11 of the switch register is 0 (down); it will be punched on the high-speed punch if bit 11 of the switch register is a 1 (up). When using the 33 ASR for symbol table output, the telepunch should be left on, since the symbol table produced may be read by DDT (see Digital-8-4-5). All diagnostics will be typed on the 33 ASR (except for the undefined address diagnostic when using the high-speed punch and the bit 11 switch option). The binary tape produced during PASS2 will be punched using the 33 ASR punch or the 75E Punch if it is included in the machining configuration and turned on. The only diagnostic in PASS2 will be Illegal Reference. Since this is typed on the 33 ASR, it may also be punched on the binary tape. It will, however, be ignored by the Binary Loader. The bit 11 switch option may be used during PASS3 also. If the machine is not equipped with the 75E High-Speed Punch, bit 11 will have no effect.

In addition to the binary tape of the Assembler, the user is provided with an ASCII tape containing symbol definitions for the instruction sets of the available options to the PDP-8 (i.e., card readers, magnetic tapes, A/D converters). Since there is only a finite amount of space available, expanding the number of permanent symbols that the Assembler recognizes decreases the maximum number of symbols the user may have available. For this reason, the ASCII Extended Definitions tape should be edited to contain definitions for only those options which the user has acquired. This tape should be read into the Assembler only on PASS1. Since it permanently fixes the symbols it contains, it should not be read again until PAL III is reloaded.

1. Load the Assembler using either the 33 ASR Reader or the 750C Photo-Electric reader.

2. Set 0200 into the switch register; press LOAD ADDRESS.

3. Place the source language tape in the reader. Turn the reader on; turn the punch on. Be certain that leader code is in the reader.
4. Set Bits \(\emptyset\) and 1 of the switch register for the proper pass. These settings are:

<table>
<thead>
<tr>
<th>Bit (\emptyset)</th>
<th>Bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PASS1 is required so that the Assembler can initialize its symbol table and define all user symbols. After PASS1 has been made, either PASS2 or PASS3 may be made.

5. Bit 11 switch option

- During PASS1 Bit 11 = 1 Punch symbol table on high-speed punch if it is in the machine configuration.
- Bit 11 = \(\emptyset\) Type (and punch) the symbol table on the 33 ASR.
- During PASS2 No effect
- During PASS3 Bit 11 = 1 Punch assembly listing tape, in ASCII, on high-speed punch.
- Bit 11 = \(\emptyset\) Type assembly listing on 33 ASR.

6. Press START. The Assembler will halt at the end of each pass. Proceed from step 3. If the Assembler has halted because of a PAUSE statement, put the next tape into the reader and press CONTINUE.

**SUMMARY**

**PASS1**

The Assembler reads the source tape, defines all user symbols, and outputs the user symbol table in alphabetic order. PASS1 diagnostics are:

- **IC** dddd AT xxxx Illegal Character

  where dddd is the value of the illegal character and xxxx is the value of the current location counter when the character was processed. The character is ignored.

- **RD** XXXXXXX AT dddd ReDefinition

  where XXXXXXX is the symbol being redefined and dddd is the value of the current location counter at the point of redefinition. The symbol is redefined.

- **DT** XXXXXXX AT dddd Duplicate Tag
An attempt is being made to redefine a symbol using the comma. XXXXXX is the symbol and dddd is the value of the current location counter. The previous value of the symbol is retained and the symbol is not redefined.

ST XXXXXX AT dddd  Symbol Table full

where XXXXXX is the symbol causing the overflow and dddd is the value of the Current Location Counter at the point of overflow. The Assembler halts and may not be restarted.

UA XXXXXX AT dddd  Undefined Address

where XXXXXX is the symbol that was used, but never defined, and dddd is the value of the Current Location Counter when the symbol was first processed. This is typed with the symbol table at the end of PASS1. The symbol is assigned a value equal to the highest address on the memory page where it was first used.

The Assembler reads the source tape and using the symbol table defined during PASS1, generates and punches the binary code. This binary tape may then be loaded by the Binary Loader. The PASS2 diagnostic is:

IR dddd AT xxxx  Illegal Reference

where dddd is the address being referenced and xxxx is the value of the Current Location Counter. The illegal address is then treated as if it were on the proper memory page.

Example:

```
*7306
JMP 307
```

would produce:

```
IR 0307 AT 7306
```

and would generate 5307 to be loaded into location 7306.
The Assembler reads the source tape and, using the symbol table defined during PASS1 generates and types the code represented by the source statements. The Current Location Counter, the contents, and the source statement are typed side by side on one line. If bit 11 of the switch register is a 1 and the machine configuration includes the high-speed punch, the assembly listing will be punched in ASCII. The PASS3 diagnostic is Illegal Reference.
chapter 6

symbol table alteration

pal iii contains a table of symbol definitions for the basic pdp-8 and its most common optional peripheral devices. these are the symbols such as tad, rfc or spa, which do not have to be defined in every program. this table is considered to be pal iii's permanent symbol table. all the symbols it contains are listed under the heading basic symbols in appendix 1 of this manual. if the user had purchased one or more of the optional devices whose instruction set is not defined among the basic symbols, for example, eaе or an a/d converter, it would be desirable if he could add the necessary symbol definitions to the permanent symbol table. this would eliminate the need for him to define these symbols in every program he writes. the opposite case would be the user who needs more space for his symbols. he would like to be able to delete all definitions except the ones he will actually use in his program.

for such purposes pal iii has three pseudo-instructions that may be used to alter its permanent symbol table. these pseudo-instructions are recognized by the assembler only during pass1. during either pass2 or 3, they are ignored and have no effect.

the pseudo-instructions that alter the symbol table are:

expunge expunge the entire permanent symbol table, except for the 9 pseudo-instructions listed in appendix 1 under basic symbols.

fixmri fix memory reference instructions. this must be followed on the same line by a symbol definition statement (parameter assignment) since the memory reference instructions are constructed in the symbol immediately following the pseudo-instructions. in other words the letters fixmri must be followed by one space, the symbol for the mri to be defined, an equal sign, and the actual value of the symbol to the immediate left of the equal sign. the pseudo-instruction must be repeated for each mri to be defined. all mri's must be defined before the definition of any other symbol.

example: expunge
fix mri tad = 1000
fix mri dca = 3000

6-1
**FIXTAB**  
FIX the current symbol **TABLE**. All symbols that have been defined before the occurrence of this pseudo-instruction are made part of the permanent symbol table and will not be printed in the symbol table at the end of PASS1 or PASS3.

An actual tape to add two symbols to those already in PAL III's permanent symbol table would have punched on it in ASCII:

```
CDF=6201
CIF=6202
FIXTAB
PAUSE
```

To use such a tape the user would:

1. Read in PAL III with the Binary Loader.
2. Set 200 in the SWITCH REGISTER and press LOAD ADDRESS.
4. Put definitions tape (ASCII) in the proper reader.
5. Press START.

The **PAUSE** pseudo-instruction at the end of the tape indicates to the Assembler that the current PASS is not ended and another tape is to follow.

6. With switches still set to PASS1, put user's program in reader and press CONTINUE on the console.

The next program to be assembled should not be preceded by the definitions since they are already in the permanent symbol table and will be there until PAL III is reloaded.

After altering the symbol table to fit his needs the user might wish to keep PAL III in this state. This can be done by punching a binary of the section of core occupied by PAL with its new symbol table.

To do this:

1. Read in PAL III and modify symbol table as desired.
2. PAL III's symbol table begins at location 2350\(_8\). Count all the symbols in the altered symbol table. Since each symbol and its value require four registers, multiply this number by 4. Convert this number to octal and add it to 2350\(_8\). This number is the upper limit of PAL III. The lower limit is 0001.
3. Using the directions for Binary Punch Routine. (Digital-8-5-U) and the limits as stated in 2 above punch out the PAL III Assembler itself.

4. The output of the Binary Punch Routine is the Assembler with the modified Symbol Table and may be loaded with the binary loader.

EXAMPLE: PAL III is loaded.

The following ASCII tape is read in on PASS1:

CDF = 6201
CIF = 6202
RDF = 6214
RIJF = 6224
RMF = 6244
RIB = 6234
FIXTAB
PAUSE

The Assembler now has in its symbol table the "MEMORY EXTENSION CONTROL" symbols and definitions. Six symbols were added and none removed. There were 84 symbols in the basic Assembler, there are now 90 symbols which require a total of 360 (10) or 550, locations. Since the symbol table starts at 2350, it extends to 2350 + 550 or 3120. The Binary Punch Routine is used to punch from 0001 through 3120 and the output is the Assembler with all the basic symbols plus memory extension symbols.
# APPENDIX 1

## SYMBOL LISTS

### BASIC SYMBOLS

<table>
<thead>
<tr>
<th>PSEUDO INSTRUCTIONS</th>
<th>FLOATING-POINT INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD</td>
<td>FEXT 00000</td>
</tr>
<tr>
<td>EXPUNGE</td>
<td>FADD 10000</td>
</tr>
<tr>
<td>FIXMRI</td>
<td>FSUB 20000</td>
</tr>
<tr>
<td>PAUSE</td>
<td>FMPY 30000</td>
</tr>
<tr>
<td>FIXTAB</td>
<td>FDIV 40000</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>FGET 50000</td>
</tr>
<tr>
<td>OCTAL</td>
<td>FPUT 60000</td>
</tr>
<tr>
<td>I</td>
<td>FNOR 70000</td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

### MEMORY REFERENCE INSTRUCTIONS

<table>
<thead>
<tr>
<th>AND 0000</th>
<th>FLOATE-POINT INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAD 1000</td>
<td>FEXT 00000</td>
</tr>
<tr>
<td>ISZ 2000</td>
<td>FADD 10000</td>
</tr>
<tr>
<td>DCA 3000</td>
<td>FSUB 20000</td>
</tr>
<tr>
<td>JMS 4000</td>
<td>FMPY 30000</td>
</tr>
<tr>
<td>JMP 5000</td>
<td>FDIV 40000</td>
</tr>
<tr>
<td></td>
<td>FGET 50000</td>
</tr>
<tr>
<td></td>
<td>FPUT 60000</td>
</tr>
<tr>
<td></td>
<td>FNOR 70000</td>
</tr>
</tbody>
</table>

### PROGRAM INTERRUPT

| ION 001 | |
| IOF 002 | |

### HIGH-SPEED READER

| RSF 6011 |
| RRB 6012 |
| RFC 6014 |

### HIGH-SPEED PUNCH

| PSF 6021 |
| PCF 6022 |
| PPC 6024 |
| PLS 6026 |

### KEYBOARD/READER

| KSF 6031 |
| KCC 6032 |
| KRS 6034 |
| KRB 6036 |

### GROUP 2 OPERATES

| HLT 7402 |
| OSR 7404 |

### COMBINED OPERATES

| CIA 7941 |
| LAS 7604 |

A1-1
SKP 7410  STL 7120
SNL 7420  GLK 7204
SZL 7430  STA 7240
SZA 7440
SNA 7450
SMA 7500
SPA 7510

/DECTAPE DUAL TRANSPORT TYPE 555 AND CONTROL TYPE 552

MMMM 6757  MMSF 6761
MMMF 6756  MMCF 6772
MMML 6766  MMSC 6771
MMLS 6751  MMRS 6774
MMLM 6752  MMCC 6762
MMLF 6754  MMLC 6764

/DECTAPE TRANSPORT TYPE TU55 AND CONTROL TYPE TC01

DTRA 6761  DTSF 6771
DTCA 6762  DTRB 6772
DTXA 6764  DTLB 6774

/MEMORY PARITY TYPE 188

SMP 6101
CMP 6104

/EXTENDED SYMBOLS

/PDP -5 EAE SYMBOLS 153*

CAM 6101  SZO 6114
LMQ 6102  DIV 6121
LAR 6104  RDM 6122
MUL 6111  SAF 6124
RDA 6112

/PDP-8 EAE SYMBOLS 182

MUY 7405  ASR 7415
DVI 7407  LSR 7417
NMI 7411  MQL 7421
SHL 7413  SCA 7441
MQA 7501  CAM 7621

/MEMORY EXTENSION CONTROL TYPE 183

CDF 6201  RIF 6224
CIF 6202  RMF 6244
RDF 6214  RIB 6234

/AUTO RESTART TYPE KR01

SPL = 6102

* PDP-5 EAE symbol definitions do not appear on the actual tape due to a conflict in the CAM instructions of PDP-5 and PDP-8. PDP-8 EAE symbols should be deleted if those for PDP-5 are inserted in the extended symbols tape.
AD CONVERTER TYPE 189

ADC 6084

AD CONVERTER/MULTIPLEXER 138E/139E

ADSF 6531    ADCC 6541
ADCV 6532    ADSC 6542
ADR 6534     ADIC 6544

OSCILLOSCOPE DISPLAY TYPE 34D

DCX 6051     DYL 6063
DXL 6053     DIX 6054
DCY 6061     DIY 6064
DXS 6057     DYS 6067

SCOPE TYPE 30N

DLB 6074

LIGHT PEN TYPE 37Ω

DSF 6071     DCF 6072

PLOTTER AND CONTROL TYPE 35Ω

PLSF 6501    PLCF 6502
PLPU 6504    PLPR 6511
PLPV 6512    PLDD 6514
PLPL 6521    PLUD 6522
PLPD 6524

CARD READER AND CONTROL TYPE CR01C

RCSF 6631    RCSP 6671
RCRA 6632    RCSE 6671
RCRB 6634    RCRD 6674

CARD READER TYPE 451

CRSF 6632    CERS 6634
CRRB 6671    CRSA 6672
CRSB 6674

CARD PUNCH AND CONTROL TYPE 45Ω

CPSF 6631    CPSE 6642
CPCF 6641    CPLB 6644
                /CERS as appears under card reader 451

LINE PRINTER TYPE 645

LCF 6652     LPR 6655
LSF 6661     LCB 6662
LLB 6664

A1-3
/SERIAL DRUM 25Ø AND 251

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRCR</td>
<td>6603</td>
<td>DRCW</td>
<td>6605</td>
</tr>
<tr>
<td>DRCF</td>
<td>6611</td>
<td>DREF</td>
<td>6612</td>
</tr>
<tr>
<td>DRTS</td>
<td>6615</td>
<td>DRSE</td>
<td>6621</td>
</tr>
<tr>
<td>DRSC</td>
<td>6622</td>
<td>DRCN</td>
<td>6624</td>
</tr>
</tbody>
</table>

/MAGNETIC TAPE TYPE 57Ø

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSCR</td>
<td>6701</td>
<td>MCD</td>
<td>6702</td>
</tr>
<tr>
<td>MTS</td>
<td>6706</td>
<td>MSUR</td>
<td>6711</td>
</tr>
<tr>
<td>MNC</td>
<td>6712</td>
<td>MTC</td>
<td>6716</td>
</tr>
<tr>
<td>MSWF</td>
<td>6721</td>
<td>MDWF</td>
<td>6722</td>
</tr>
<tr>
<td>MCWF</td>
<td>6722</td>
<td>MEWF</td>
<td>6722</td>
</tr>
<tr>
<td>MWF</td>
<td>6722</td>
<td>MSEF</td>
<td>6731</td>
</tr>
<tr>
<td>MDEF</td>
<td>6732</td>
<td>MCED</td>
<td>6732</td>
</tr>
<tr>
<td>MEIF</td>
<td>6732</td>
<td>MIEF</td>
<td>6732</td>
</tr>
<tr>
<td>MTRS</td>
<td>6734</td>
<td>MCC</td>
<td>6741</td>
</tr>
<tr>
<td>MRWC</td>
<td>6742</td>
<td>MRCA</td>
<td>6744</td>
</tr>
<tr>
<td>MCA</td>
<td>6745</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/MAGNETIC TAPE TYPE 58Ø

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSRD</td>
<td>6715</td>
<td>TIFM</td>
<td>6707</td>
</tr>
<tr>
<td>TSWR</td>
<td>6716</td>
<td>TSDF</td>
<td>6721</td>
</tr>
<tr>
<td>TSSR</td>
<td>6722</td>
<td>TSST</td>
<td>6724</td>
</tr>
<tr>
<td>TWRT</td>
<td>6731</td>
<td>TCPI</td>
<td>6732</td>
</tr>
<tr>
<td>TSRS</td>
<td>6734</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/EIGHT CHANNEL SAMPLE AND HOLD CONTROL TYPE ACØ1A

/OPTION TO TYPE 139E MULTIPLEXOR

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSC</td>
<td>6571</td>
</tr>
<tr>
<td>HAC</td>
<td>6572</td>
</tr>
<tr>
<td>SAC</td>
<td>6574</td>
</tr>
</tbody>
</table>

/DATA COMMUNICATION SYSTEMS TYPE 63Ø

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTINCR</td>
<td>6401</td>
<td>TTRL</td>
<td>6414</td>
</tr>
<tr>
<td>TTI</td>
<td>6402</td>
<td>TTSKP</td>
<td>6421</td>
</tr>
<tr>
<td>TTO</td>
<td>6404</td>
<td>TTXON</td>
<td>6422</td>
</tr>
<tr>
<td>TCL</td>
<td>6411</td>
<td>TTXOF</td>
<td>6424</td>
</tr>
<tr>
<td>TSL</td>
<td>6412</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX 2

**ASCII CHARACTER SET**

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>301</td>
<td>Ø</td>
<td>260</td>
</tr>
<tr>
<td>B</td>
<td>302</td>
<td>1</td>
<td>261</td>
</tr>
<tr>
<td>C</td>
<td>303</td>
<td>2</td>
<td>262</td>
</tr>
<tr>
<td>D</td>
<td>304</td>
<td>3</td>
<td>263</td>
</tr>
<tr>
<td>E</td>
<td>305</td>
<td>4</td>
<td>264</td>
</tr>
<tr>
<td>F</td>
<td>306</td>
<td>5</td>
<td>265</td>
</tr>
<tr>
<td>G</td>
<td>307</td>
<td>6</td>
<td>266</td>
</tr>
<tr>
<td>H</td>
<td>310</td>
<td>7</td>
<td>267</td>
</tr>
<tr>
<td>I</td>
<td>311</td>
<td>8</td>
<td>270</td>
</tr>
<tr>
<td>J</td>
<td>312</td>
<td>9</td>
<td>271</td>
</tr>
<tr>
<td>K</td>
<td>313</td>
<td>$</td>
<td>244</td>
</tr>
<tr>
<td>L</td>
<td>314</td>
<td>*</td>
<td>252</td>
</tr>
<tr>
<td>M</td>
<td>315</td>
<td>+</td>
<td>253</td>
</tr>
<tr>
<td>N</td>
<td>316</td>
<td>,</td>
<td>254</td>
</tr>
<tr>
<td>O</td>
<td>317</td>
<td>-</td>
<td>255</td>
</tr>
<tr>
<td>P</td>
<td>320</td>
<td>.</td>
<td>256</td>
</tr>
<tr>
<td>Q</td>
<td>321</td>
<td>/</td>
<td>257</td>
</tr>
<tr>
<td>R</td>
<td>322</td>
<td>;</td>
<td>273</td>
</tr>
<tr>
<td>S</td>
<td>323</td>
<td>=</td>
<td>275</td>
</tr>
<tr>
<td>T</td>
<td>324</td>
<td>Space</td>
<td>240</td>
</tr>
<tr>
<td>U</td>
<td>325</td>
<td>Tab</td>
<td>211</td>
</tr>
<tr>
<td>V</td>
<td>326</td>
<td>Line Feed</td>
<td>212</td>
</tr>
<tr>
<td>W</td>
<td>327</td>
<td>Form Feed</td>
<td>214</td>
</tr>
<tr>
<td>X</td>
<td>330</td>
<td>Carriage-Return</td>
<td>215</td>
</tr>
<tr>
<td>Y</td>
<td>331</td>
<td>Rubout</td>
<td>377</td>
</tr>
<tr>
<td>Z</td>
<td>332</td>
<td>Leader/Trailer</td>
<td>200*</td>
</tr>
</tbody>
</table>

*Code 200* may be used as leader/trailer. It is generated by depressing:

Shift, CTRL, Repeat, @

Release the keys in reverse order.
NOTE 1: PAL III does not require the presence of Channel 8. Thus, 181 is considered equivalent to 301. This is useful if the paper tape is prepared on a Teletype that punches parity.

NOTE 2: All other characters are valid within comments.