intel®

8080
PL/M™ Compiler
Operators Manual

REVISION A

98-103A

PREFACE

This manual describes the operation of the INTEL 8080 PL/M* Cross Compiler. The compiler comprises two distinct programs written in ANSI standard FORTRAN IV and may be installed on most medium to large scale computer systems. Some details presented in this manual may vary due to system dependencies and compiler options selected during the installation process. The PL/M language itself is described in the 8008 and 8080 PL/M Programming Manual.

* PL/M is a claimed trademark of INTEL Corporation

CONTENTS

1. INTRODUCTION

The 8080 PL/M Cross Compiler comprises two distinct programs which must be executed consecutively to perform a complete compilation of a PL/M source program. The two programs are known as Pass 1 and Pass 2 of the PL/M Compiler, and are sometimes referred to as PLM81 and PLM82 respectively.

The first pass reads a PL/M source program and converts it to an intermediate form on work files. Optionally, a listing of the input source program may be obtained during this pass. Errors in program syntax are detected at this stage, and appropriate error messages are sent to the list file.

The second pass of the PL/M Compiler processes the intermediate files created by Pass 1, and generates the machine code for the MCS-80 CPU. This machine code, which may be in either BNPF or Hex format, may be loaded and executed directly on an INTELLEC ® 8/Mod 80 Microcomputer Development System, or simulated using INTERP/80, a cross-simulator of the 8080 CPU. It may also be used for the programming of ROMs. Pass 2 of the compilation process will produce, optionally, a symbol table, and mnemonic listing of the generated machine code. Certain errors may be detected during this phase, and these are also reported in the list file.

Figure 1 illustrates the overall file structure and flow of program execution of the PL/M Compiler. The reader may find it helpful to refer to this diagram as he reads subsequent sections of this manual.

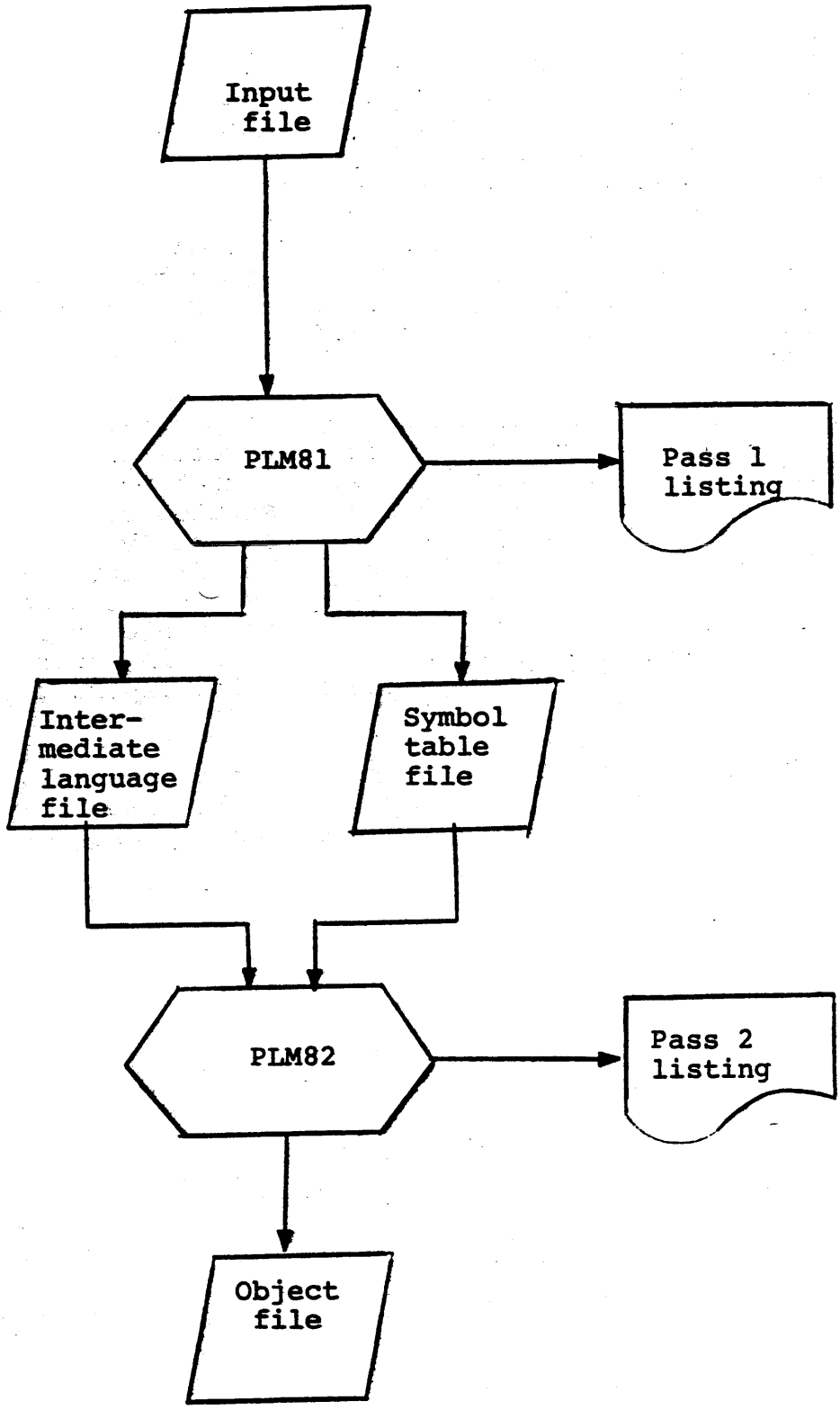

Figure 1

File structure and flow of program execution

2. COMPILER CONTROLS

The operation of each pass of the PL/M Compiler is governed by a set of parameters known as compiler controls, each control being identified by a unique letter of the alphabet. Compiler controls may perform one of three functions, as follows:

a) Definition of the characteristics of the files accessed by the PL/M Compiler, such as FORTRAN unit number, and maximum record size.

b) Selection of those optional features of the compiler which are to be invoked during a particular compilation.

c) Specification of various compile-time parameters, such as the location of the first pages of RAM and ROM in the object system.

In general, each compiler control contains a non-negative integer value, although some controls are restricted to the values 1 and 0 indicating an 'on' or 'off' condition, respectively. Each compiler control is provided with a default value which it assumes throughout the compilation process unless explicitly altered by the user. Appendix B provides a complete list of the controls available in both Pass 1 and Pass 2 of the PL/M compiler, along with their default values. In practice, however, some of these defaults may have been changed during installation of the PL/M Compiler on any specific system. Further details must be obtained from your local programming or time-sharing staff.

The value associated with any particular compiler control may be changed at any stage of the compilation process by the compiler user. This is accomplished by a line of input with a dollar sign ($) in the first character position processed. Thus the PL/M Compiler distinguishes between lines that belong to the PL/M source program proper and lines that change the values of compiler controls by the absence or presence, respectively, of a dollar sign in the first character position. Such lines which alter the values of compiler controls are known as control records.

Each control record may respecify the values of a number of compiler controls. Each specification comprises:

a) A dollar sign (appearing in the first character position processed, for the first specification).

b) One or more letters, the first of which is that identifying the particular control (see Appendix B), and the remainder are optional and may be used for purposes of self-documentation.

c) An equals sign (=).

d) A non-negative integer.

Spaces are not permitted between the first two elements (a and b above) of a compiler control specification. They are permitted elsewhere. For example:

```
$I=6
```

is a control record which changes the value of the I control to 6.

Several control specifications may appear in the same control record. Each follows the format defined above, and is separated from the next by zero or more space characters. For example:

```
$I=6 $O=2
```

is a control record which respecifies the values of the I and O controls. Note that the new values of the controls do not become operative until the whole record has been processed.

Two special specification formats are available which permit the user to interrogate the current values of the compiler controls. Two consecutive dollar signs ($$) appearing on their own cause the compiler to list the values of all compiler controls. A double dollar sign appearing in front of a control letter causes the value of that control to be displayed. For example, the control record

```
$INPUT=6$OUTPUT=2$$INPUT$$
```

causes the I and O controls to assume the values 6 and 2, respectively. The value of the I control is then displayed, in this case 6, followed by the current values of all the compiler controls.

The user should be aware that only a small number of compiler controls may require explicit setting during a particular compilation. Many will usually be supplied with a permanent value during system installation, while others control diagnostic features useful only in the event of compiler failure.

3. FILE SYSTEM

All input and output performed by the PL/M Compiler is specified in terms of 'FORTRAN Units', which are defined as part of ANSI Standard FORTRAN and provide a machine independent file addressing scheme.

Each FORTRAN unit is identified by a unique integer; for example, FORTRAN Unit 5, FORTRAN Unit 20. The compiler user directs input and output to specific FORTRAN units by the use of certain compiler controls. Each value of such a control uniquely specifies a FORTRAN unit. For example, in Pass 1 the I control defines the unit supplying source input. The mapping between control values and FORTRAN unit numbers is defined in Appendix C.

In any particular installation, each FORTRAN unit will correspond either to an input/output device, such as a teletype, card reader or line printer; or a disk file identified by a specific file name. This manual does not specify the device type or file name corresponding to each FORTRAN unit. This information will be local to any given implementation of the compiler. However, Appendix D provides file definitions for the versions of the compiler available from Tymshare, General Electric, and United Computing Systems. Section 5 of this manual contains an example of compiler operation on a PDP-10, and typical PDP-10 file names have been specified.

4. COMPILER OPERATION

A complete compilation of a PL/M program is performed in two distinct phases, known as Pass 1 and Pass 2. The operation of each of these phases is discussed separately.

4.1 Pass 1

The first pass of the PL/M Compiler reads a PL/M source program, and generates two intermediate files (an encoded symbol table, and an intermediate language) which may subsequently be processed by Pass 2. A listing of the source program may be produced during Pass 1, depending on whether the P compiler control is set 'on' or 'off' (i.e. has a value of 1 or 0, respectively). Each line of the source file listing comprises three elements:

a) Line number
b) Current level of nesting of the PL/M source.
c) An echo of the input source record.

Error messages may also be produced during Pass 1 of the compilation process. These take the form:

(nnnnn) ERROR m NEAR s

where nnnnn specifies the line number on which the error occurred, s is a symbol on the line near the error, and m specifies an error code which may be interpreted by reference to Appendix A.

The operation of Pass 1 begins by taking input from the file specified by the default value of the I compiler control. This may be a card reader, or interactive terminal, for example, depending on the system configuration established during the installation of the compiler. The compiler continues to accept input (both PL/M source and control records) from this file until an EOF token is detected, indicating end-of-file, or until the value of the I control is respecified by a control record. In the latter case, input switches to the new file which has been specified. In this way, Pass 1 can compile a PL/M program which has been previously created, and is resident in a file on disk.

Other controls commonly used during execution of Pass 1 are as follows:

L Left margin. This control specifies the first character position of each input record to be processed during compilation. It might be used for example to instruct the compiler to ignore any sequence numbers which appear in the first few character positions of each record in the file.

R Right margin. This control is similar to L, but specifies a right margin.

O Output. This control is similar to the I control but specifies the file which is to receive the program listing and error messages. For example, in some installations this could be the terminal, or perhaps a designated disk file.

4.2 Pass 2

Pass 2 performs the second phase of a PL/M compilation by processing the two intermediate files generated by Pass 1. The operation of Pass 2 begins by accepting input from the file specified by the default value of the I compiler control. Normally, this would correspond to the terminal in an interactive environment, or card reader in a batch environment. At this stage any number of control records may be input to set up the desired values of Pass 2 compiler controls. The end of such input is signalled by a special record containing zero or more space characters only. (For example: a blank card if in batch mode, or an extra carriage-return from a terminal.) When the special blank record is encountered, Pass 2 automatically processes the intermediate files and generates the MCS-80 object code.

Pass 2 generates a list file which contains the output of the various compiler options - e.g. a symbol table. These options are discussed later with their respective controls. Pass 2 may also report on error conditions in a manner similar to Pass 1, and appropriate error messages will appear in the list file. A complete list of Pass 2 error codes is given in Appendix A. The user should be aware that Pass 2 errors may arise from a number of possible causes:

a) A source program error undetected by Pass 1.
b) A compiler installation problem, or misoperation of the compiler.
c) Compiler failure - for example, an internal table overflow.

Compiler controls commonly used during Pass 2 are as follows:

F If set, Pass 2 generates a decoded representation of the object code produced.

G If set, Pass 2 generates a table indicating the approximate location in memory of the code produced by each line of PL/M source.

H The address, in decimal, of the start of the object code in memory.

M If set, Pass 2 generates a symbol table in the list file.

O File number of the list file (see Pass 1).

Q If Q is zero, the object file is in BNPF format. If Q is nonzero, the object file is in Hex format.

V The number of the first page to be allocated as variable storage. If V is zero, the allocation is made automatically.

5. SAMPLE EXECUTION OF THE PL/M COMPILER

The exact manner in which PLM81 and PLM82 operate on any particular computer is implementation dependent. Figure 5-1 gives a step-by-step example of the operation of both passes of the compiler on a PDP-10 computer system. Figure 5-2 shows the file structure and the flow of the program execution. File names are specified for each of the files accessed.

Using this version, for example, the programmer places the PL/M source program into a file named FOR20.DAT, which corresponds to the file referenced by a value of 6 in the I compiler control. This file is read when a $I=6 control record is encountered during PLM81 execution. PLM81 produces the intermediate files FOR22.DAT and FOR23.DAT, along with a spooled source file listing, by setting $O=2. The output of Pass 1 is shown in Figure 5-3.

PLM82 is then initiated to process the intermediate files produced by PLM81. Output listing is again directed to a spooled print file using the $O=2 control. The hexadecimal object file produced by PLM82 is written to the file FOR21.DAT.

The output of the $G compiler option is illustrated by Figure 5-4. It comprises a table whose entries each have two numbers, separated by an equals sign. To the left of an equals sign is a source line number, and to the right is the approximate location, in hexadecimal, of the object code generated by the specified source line.

The output of the $M compiler option is illustrated by Figure 5-5. It comprises a table of PL/M identifiers (variables, labels, and procedures) in order of their appearance in the PL/M source, along with their assigned locations in memory (in hexadecimal).

The output of the $F compiler control is illustrated by Figure 5-6. The left hand column of the table identifies locations in memory, and the entries in the remainder of each row indicate the initial contents of the designated memory locations. Operation codes are expressed in a mnemonic format, similar to the mnemonics accepted by the 8080 assembler. Other locations, for example those initialized by a DATA statement or INITIAL attribute, are expressed as hexadecimal numbers.

Figure 5-7 shows the hexadecimal object file produced by Pass 2. This comprises two sections:

a) A symbol map consisting of symbol numbers, names, and hexadecimal memory locations. This map is used by the 8080 cross simulator to provide symbolic debugging facilities.
b) Hexadecimal object code in standard 8080 format.

```
.COPY FOR20.DAT=MYPROG.PLM                    (1)

.RUN PLM81                                    (2)

8080 PLM1 VERS 2.0                            (3)

$O=2 $I=6                                     (4)

NO PROGRAM ERRORS                             (5)

.RUN PLM82                                    (6)

8080 PLM2 VERS 2.0                            (7)

$O=2 $F=1 $G=1                                (8)

                                              (9)
-
NO PROGRAM ERRORS                             (10)

.PRINT *.LPT                                  (11)

.PUNCH FOR21.DAT                              (12)
```

(1) Copy the source program into file FOR20.DAT from file MYPROG.PLM.
(2) Invoke Pass 1 of the PL/M Compiler.
(3) PL/M Compiler types its identity.
(4) Divert input to file number 6, which corresponds to FOR20.DAT. Divert output to file number 2, the spooled list file.
(5) Pass 1 types an error summary.
(6) Invoke Pass 2 of the PL/M Compiler.
(7) Pass 2 types its identity.
(8) Divert output to a spooled list file, and select the F and G compiler options.
(9) Blank line starts Pass 2 compilation process.
(10) Pass 2 types an error summary.
(11) The spooled list files are printed.
(12) The Pass 2 Hex output is punched, for subsequent loading to an INTELLEC 8/MOD 80 Microcomputer Development System.

Note: Underlined commands are those typed by the user.

Figure 5-1 Compiler Operation on a PDP-10

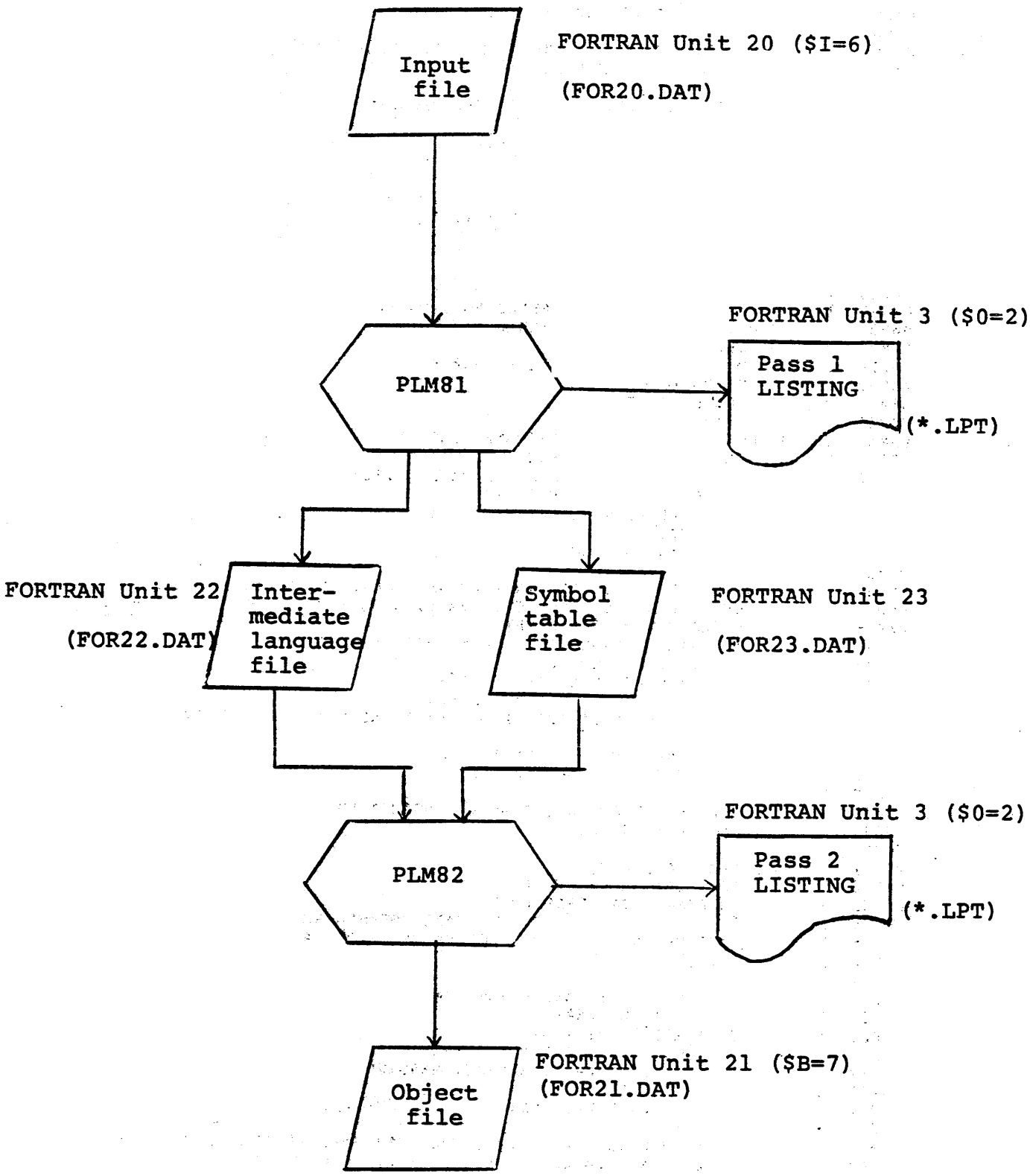

Figure 5-2

File structure and flow of program execution on a PDP-10

```
00001  1   /*
00002  1                              SAMPLE  PL/M  PROGRAM
00003  1
00004  1       THIS PROGRAM CALCULATES AND PRINTS OUT THE SQUARE ROOTS OF
00005  1       ALL INTEGERS BETWEEN 1 AND 1000.
00006  1                                                                  */
00007  1   DECLARE CR LITERALLY '0DH', LF LITERALLY '0AH', TRUE LITERALLY '1',
00008  1           FALSE LITERALLY '0';
00009  1
00010  1   100H:  /*  IS THE ORIGIN OF THIS PROGRAM */
00011  1
00012  1   SQUARE$ROOT: PROCEDURE(X) BYTE;
00013  2       DECLARE (X,Y,Z) ADDRESS;
00014  2       Y=X; Z=SHR(X+1,1);
00015  2           DO WHILE Y<>Z;
00016  2           Y=Z; Z=SHR(X/Y + Y + 1, 1);
00017  3           END;
00018  2       RETURN Y;
00019  2       END SQUAREROOT;
00020  1
00021  1       /* PRINT USING INTELLEC MONITOR */
00022  1   PRINT$CHAR: PROCEDURE (CHAR);
00023  2       DECLARE CHAR BYTE;
00024  2       DECLARE IOCO LITERALLY '3809H';
00025  2       GO TO IOCO;
00026  2       END PRINT$CHAR;
00027  1
00028  1   PRINT$STRING: PROCEDURE(NAME,LENGTH);
00029  2       DECLARE NAME ADDRESS,
00030  2           (LENGTH,I,CHAR BASED NAME) BYTE;
00031  2           DO I = 0 TO LENGTH-1;
00032  2           CALL PRINT$CHAR(CHAR(I));
00033  3           END;
00034  2       END PRINT$STRING;
00035  1
00036  1   PRINT$NUMBER: PROCEDURE(NUMBER,BASE,CHARS,ZERO$SUPPRESS);
00037  2       DECLARE NUMBER ADDRESS, (BASE,CHARS,ZERO$SUPPRESS,I,J) BYTE;
00038  2       DECLARE TEMP(16) BYTE;
00039  2       IF CHARS > LAST(TEMP) THEN CHARS = LAST(TEMP);
00040  2           DO I = 1 TO CHARS;
00041  2           J=NUMBER MOD BASE + '0';
00042  3           IF J > '9' THEN J = J + 7;
00043  3           IF ZERO$SUPPRESS AND I <> 1 AND NUMBER = 0 THEN
00044  3               J = ' ';
00045  3           TEMP(LENGTH(TEMP)-I) = J;
00046  3           NUMBER = NUMBER / BASE;
00047  3           END;
00048  2       CALL PRINT$STRING(.TEMP + LENGTH(TEMP) - CHARS,CHARS);
00049  2       END PRINT$NUMBER;
00050  1
00051  1   DECLARE I ADDRESS,
00052  1       CRLF LITERALLY 'CR,LF',
00053  1       HEADING DATA (CRLF,LF,LF,
00054  1                     '                TABLE OF SQUARE ROOTS', CRLF,LF,
00055  1       ' VALUE  ROOT VALUE  ROOT VALUE  ROOT VALUE  ROOT VALUE  ROOT',
00056  1       CRLF,LF);
00057  1
00058  1       /* SILENCE TTY AND PRINT COMPUTED VALUES */
00059  1       DO I = 1 TO 1000;
00060  1       IF I MOD 5 = 1 THEN
00061  2           DO; IF I MOD 250 = 1 THEN
00062  3               CALL PRINT$STRING(.HEADING,LENGTH(HEADING));
00063  3               ELSE
00064  3               CALL PRINT$STRING(.(CR,LF),2);
00065  3           END;
00066  2       CALL PRINT$NUMBER(I,10,6,TRUE /* SUPPRESS LEADING ZEROES */);
00067  2       CALL PRINT$NUMBER(SQUARE$ROOT(I), 10,6, TRUE);
00068  2       END;
00069  1
00070  1   EOF
NO PROGRAM ERRORS
```

Figure 5-3 Source Program Listing-Pass 1

```
 1=0003H   12=0013H   13=0016H   14=001CH   15=002FH   16=0045H
17=00B1H   18=00BAH   19=00C0H   23=00C4H   25=00C7H   26=00C8H
29=00D0H   31=00D3H   32=00DEH   33=00EAH   34=00F1H   35=00F2H
37=00F8H   39=00FCH   40=00FFH   41=010FH   42=012CH   43=0139H
44=0156H   45=015AH   46=016AH   47=0186H   48=0191H   49=01AAH
50=01ABH   56=021EH   59=0226H   60=0235H   61=0251H   62=0270H
63=027BH   64=027EH   65=0288H   66=0292H   67=029DH   68=02B7H
69=02C6H
```

Figure 5-4 Source Line Number-Code Location
Cross Reference Listing-Pass 2

```
MEMORY.....................0300H
SQUAREROOT.................0016H
X..........................02DAH
Y..........................02DCH
Z..........................02DEH
PRINTCHAR..................00C0H
CHAR.......................02E1H
PRINTSTRING................00C8H
NAME.......................02E2H
LENGTH.....................02E4H
I..........................02E5H
PRINTNUMBER................00F2H
NUMBER.....................02E6H
BASE.......................02E9H
CHARS......................02EAH
ZEROSUPPRESS...............02EBH
I..........................02ECH
J..........................02EDH
TEMP.......................02EEH
I..........................02FEH
HEADING....................01ABH
```

Figure 5-5 Symbol Table-Pass 2

```
0010H LXI SP D4H    02H    JMP    1EH    02H    LXI H  DAH    02H
0019H MOV MC INX H  MOV MB DCR L  MOV CM INR L  MOV BM INR L  MOV MC
0022H INX H  MOV MB LHLD   DAH    02H    INX H  XCHG   MOV AD ORA A
002BH RAR    MOV DA MOV AE RAR    LXI H  DEH    02H    MOV MA INX H
0034H MOV MD LXI H  DCH    02H    MOV AM INR L  MOV BM INR L  SUB M
003DH INR L  MOV CA MOV AB SBC M  ORA C  JZ     BAH    00H    DCR L
0046H MOV CM INR L  MOV BM MOV LI DCH    MOV MC INX H  MOV MB DCR L
004FH MOV CM INR L  MOV BM MOV LI D6H    MOV MC INX H  MOV MB MOV LI
0058H DAH    MOV CM INR L  MOV BM MOV LI D8H    MOV MC INX H  MOV MB
0061H JMP    97H    00H    MOV EM DCR L  MOV DM MOV MI 11H    MOV BI
006AH 00H    MOV CB MOV AD RAL    MOV DA MOV AE RAL    MOV EM DCR E
0073H MOV ME MOV EA RZ     MOV AB RAL    MOV BA MOV AC RAL    MOV CA
007CH DCR L  DCR L  MOV AB SUB M  MOV BA INR L  MOV AC SBC M  MOV CA
0085H JNC    90H    00H    DCR L  MOV AB ADD M  MOV BA INR L  MOV AC
008EH ADC M  MOV CA INR L  SBC A  SBC I  80H    JMP    6CH    00H
0097H CALL   64H    00H    MOV LI D4H    MOV MD INR L  MOV ME LXI H
00A0H D4H    02H    MOV CM INR L  MOV BM LHLD   DCH    02H    DAD B
00A9H INX H  XCHG   MOV AD ORA A  RAR    MOV DA MOV AE RAR    LXI H
00B2H DEH    02H    MOV MA INX H  MOV MD JMP    35H    00H    MOV LI
00BBH DCH    MOV AM INR L  MOV BM RET    LXI H  E1H    02H    MOV MC
00C4H JMP    09H    38H    RET    LXI H  E2H    02H    MOV MC INX H
00CDH MOV MB INR L  MOV ME INR L  MOV MI 00H    LXI H  E4H    02H
00D6H MOV CM DCR C  MOV AC INR L  SUB M  JC     F1H    00H    MOV CM
00DFH MOV BI 00H    LHLD   E2H    02H    DAD B  MOV AM MOV CA CALL
00E8H C0H    00H    LXI H  E5H    02H    INR M  JMP    D3H    00H
00F1H RET    LXI H  EAH    02H    MOV MC INR L  MOV ME MOV AI 0FH
00FAH DCR L  SUB M  JNC    01H    01H    MOV MI 0FH    MOV LI ECH
0103H MOV MI 01H    LXI H  EAH    02H    MOV AM MOV LI ECH    SUB M
010CH JC     91H    01H    MOV LI E9H    MOV CM MOV LI D6H    MOV MC
0115H INX H  MOV MI 00H    MOV LI E6H    MOV CM INR L  MOV BM MOV LI
011EH D8H    MOV MC INX H  MOV MB CALL   64H    00H    LXI D  30H
0127H 00H    MOV LB MOV HC DAD D  XCHG   LXI H  EDH    02H    MOV ME
0130H MOV AI 39H    SUB M  JNC    3AH    01H    MOV AM ADD I  07H
0139H MOV MA DCR L  MOV CM DCR C  MOV AI FFH    JNZ    43H    01H
0142H XRA A  DCR L  ANA M  MOV LI E6H    MOV CA MOV AM INR L  MOV DM
014BH SUB I  00H    MOV EA MOV AD SBC I  00H    ORA E  SUB I  01H
0154H SBC A  ANA C  RRC    JNC    5EH    01H    MOV LI EDH    MOV MI
015DH 20H    MOV AI 10H    MOV LI ECH    SUB M  MOV CA MOV BI 00H
0166H MOV LI EEH    DAD B  XCHG   LXI H  EDH    02H    MOV CM MOV AC
016FH STAX D MOV LI E9H    MOV CM MOV LI D6H    MOV MC INX H  MOV MI
0178H 00H    MOV LI E6H    MOV CM INR L  MOV BM MOV LI D8H    MOV MC
0181H INX H  MOV MB CALL   64H    00H    MOV LI E6H    MOV MD INX H
018AH MOV ME MOV LI ECH    INR M  JMP    05H    01H    LXI B  EEH
0193H 02H    LXI D  10H    00H    MOV LC MOV HB DAD D  XCHG   MOV AE
019CH LXI H  EAH    02H    SUB M  MOV EA MOV AD SBC I  00H    MOV CE
01A5H MOV BA MOV EM CALL   C8H    00H    RET
01ABH 0DH 0AH 0AH 0AH 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H
01BBH 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 20H 54H 41H 42H 4CH
01CBH 45H 20H 4FH 46H 20H 53H 51H 55H 41H 52H 45H 20H 52H 4FH 4FH 54H
01DBH 53H 0DH 0AH 0AH 20H 56H 41H 4CH 55H 45H 20H 20H 52H 4FH 4FH 54H
01EBH 20H 56H 41H 4CH 55H 45H 20H 20H 52H 4FH 4FH 54H 20H 56H 41H 4CH
01FBH 55H 45H 20H 20H 52H 4FH 4FH 54H 20H 56H 41H 4CH 55H 45H 20H 20H
020BH 52H 4FH 4FH 54H 20H 56H 41H 4CH 55H 45H 20H 20H 52H 4FH 4FH 54H
021BH 0DH 0AH 0AH
021EH LXI H  FEH    02H    MOV MI 01H    INX H  MOV MI 00H    MOV AI
0227H E8H    MOV BI 03H    LXI H  FEH    02H    SUB M  INR L  MOV CA
0230H MOV AB SBC M  JC     C6H    02H    MOV LI D6H    MOV MI 05H
0239H INX H  MOV MI 00H    MOV LI FEH    MOV CM INR L  MOV BM MOV LI
0242H D8H    MOV MC INX H  MOV MB CALL   64H    00H    MOV AB SUB I
024BH 01H    MOV BA MOV AC SBC I  00H    ORA B  JNZ    88H    02H
0254H MOV LI D6H    MOV MI FAH    INX H  MOV MI 00H    MOV LI FEH
025DH MOV CM INR L  MOV BM MOV LI D8H    MOV MC INX H  MOV MB CALL
0266H 64H    00H    MOV AB SUB I  01H    MOV BA MOV AC SBC I  00H
026FH ORA B  JNZ    80H    02H    LXI B  ABH    01H    MOV EI 73H
0278H CALL   C8H    00H    JMP    88H    02H
027EH 0DH 0AH
0280H LXI B  7EH    02H    MOV EI 02H    CALL   C8H    00H    MOV LI
0289H FEH    MOV CM INR L  MOV BM MOV LI E6H    MOV MC INX H  MOV MB
0292H MOV LI E9H    MOV MI 0AH    MOV CI 06H    MOV EI 01H    CALL
029BH F2H    00H    MOV LI FEH    MOV CM INR L  MOV BM CALL   16H
02A4H 00H    LXI H  E6H    02H    MOV MA INX H  MOV MI 00H    MOV LI
02ADH E9H    MOV MI 0AH    MOV CI 06H    MOV EI 01H    CALL   F2H
02B6H 00H    MOV LI FEH    MOV CM INR L  MOV BM LXI H  01H    00H
02BFH DAD B  SHLD   FEH    02H    JMP    26H    02H    EI     HLT
```

Figure 5-6 Generated Object Code-Pass 2

```
   5 MEMORY 00300H
  25 SQUAREROOT 00016H
  26 X 002DAH
  28 Y 002DCH
  29 Z 002DEH
  33 PRINTCHAR 000C0H
  34 CHAR 002E1H
  37 PRINTSTRING 000C8H
  38 NAME 002E2H
  39 LENGTH 002E4H
  41 I 002E5H
  46 PRINTNUMBER 000F2H
  47 NUMBER 002E6H
  48 BASE 002E9H
  49 CHARS 002EAH
  50 ZEROSUPPRESS 002EBH
  52 I 002ECH
  53 J 002EDH
  54 TEMP 002EEH
  64 I 002FEH
  66 HEADING 001ABH
 $
*****************************************
:1000100031D402C31E0221DA027123702D4E2C4608
:10002000C7123702ADA0223EB7AB71F577B1F212A
:10003000DE0277237221DC027E2C462C962C4F7830
:1000400009EB1CABA002D4E2C462EDC7123702D4E67
:10005000C462ED67123702EDA4E2C462ED87123C4
:1000600070C397005E2D5636110600487A17577BED
:10007000175E1D735FC878174779174F2D2D789637
:10008000472C799E4FD290002D7886472C798E4F41
:1000900C2C9FDE80C36C00CD64002ED4722C7321A3
:1000A000D4024E2C462ADC020923EB7AB71F577B79
:1000B0001F21DE02772372C335002EDC7E2C46C959
:1000C00021E10271C30938C921E2027123702C7346
:1000D0002C360021E4024E0D792C96DAF1004E0602
:1000E000002AE202097E4FCDC00021E50234C3D3CD
:1000F00000C921EA02712C733E0F2D96D201013600
:10010000F2EEC360121EA027E2EEC96DA91012EBA
:10011000E94E2ED6712336002EE64E2C462ED8718F
:10012000237ACD64001130006861I9EB21ED02737A
:10013000E3996D23A017EC007772D4E0D3EFFC25C
:10014000430IAF2DA62EE64F7E2C56D6005F7ADEF9
:10015000B3D6019FA10FD25E012EED36203E10D6
:10016000EEC964F06002EEE09EB21ED024E791291
:10017000EE94E2ED6712336002EE64E2C462ED872
:10018000712370CD64002EE67223732EEC34C30508
:10019000010IEE02111000696019EB7B21EA029661
:1001A0005F7ADE004B475ECDC800C90D0A0A0A20FF
:1001B00020202020202020202020202020202020203F
:1001C000202020202020205441424C45204F462012
:1001D000535155415245205240F54530D0A0A2056
:1001E00056414C55452020524F4F542056414C55B6
:1001F000452020524F4F542056414C554520205207
:10020000F4F542056414C55452020524F4F5420BB
:1002100056414C55452020524F4F540D0A0A21FE9D
:10022000023601233600EE8060321FE02962C4FDB
:10023000789EDAC6022ED636052336002EFE4E2CC8
:10024000462ED8712370CD640078D6014779DE0040
:10025000B0C288022ED636FA2336002EFE4E2C4629
:1002600002ED8712370CD640078D6014779DE00B0B6
:10027000C2800201AB011E73CDC800C388020D0A03
:1002800017E021E02CDC8002EFE4E2C462EE671C7
:10029000237302EE9360A0E061E01CDF2002EFE4E08
:1002A0002C46CD160021E602772336002EE9360AC9
:1002B0000E061E01CDF2002EFE4E2C4621010000935
:0802C00022FE02C32602FB76B8
:00000000
*****************************************
 $
```

Figure 5-7 Hexadecimal Object Code File-Pass 2

6. RUN-TIME CONVENTIONS

This section presents the run-time organization of PL/M programs, including storage allocation and subroutine linkage, in an 8080 CPU environment.

6.1 Storage Allocation

The organization of memory for a PL/M object program is shown in Figure 6-1. Memory is allocated in three sections:

1. Instruction Storage Area (ISA)
2. Variable Storage Area (VSA)
3. Free Storage Area (FSA)

The ISA is occupied by the machine code generated by the PL/M source, and variables declared in DATA declarations.

The VSA is located above the ISA, and contains (in order of decreasing address):

1. Variables, other than DATA variables, declared in the PL/M source. They are arranged in order of declaration. ADDRESS variables are aligned on an even-byte boundary. BYTE variables are not aligned.

2. Compiler generated temporaries i.e. workspace used by the object program, but not explicitly declared.

3. The stack. The size of the stack area is determined by the compiler, unless explicit overrides are used. (See 6.4).

The compiler will normally locate the VSA directly above the ISA. However the compiler user may specify the first page of the VSA explicitly, using the pass 2 $V compiler control. (A page contains 256 bytes). This may be used, for example, to ensure that the VSA is located in RAM for a system that has both RAM and ROM.

FSA is the area of memory above the VSA. The built-in PL/M identifier MEMORY may be used to reference the FSA.

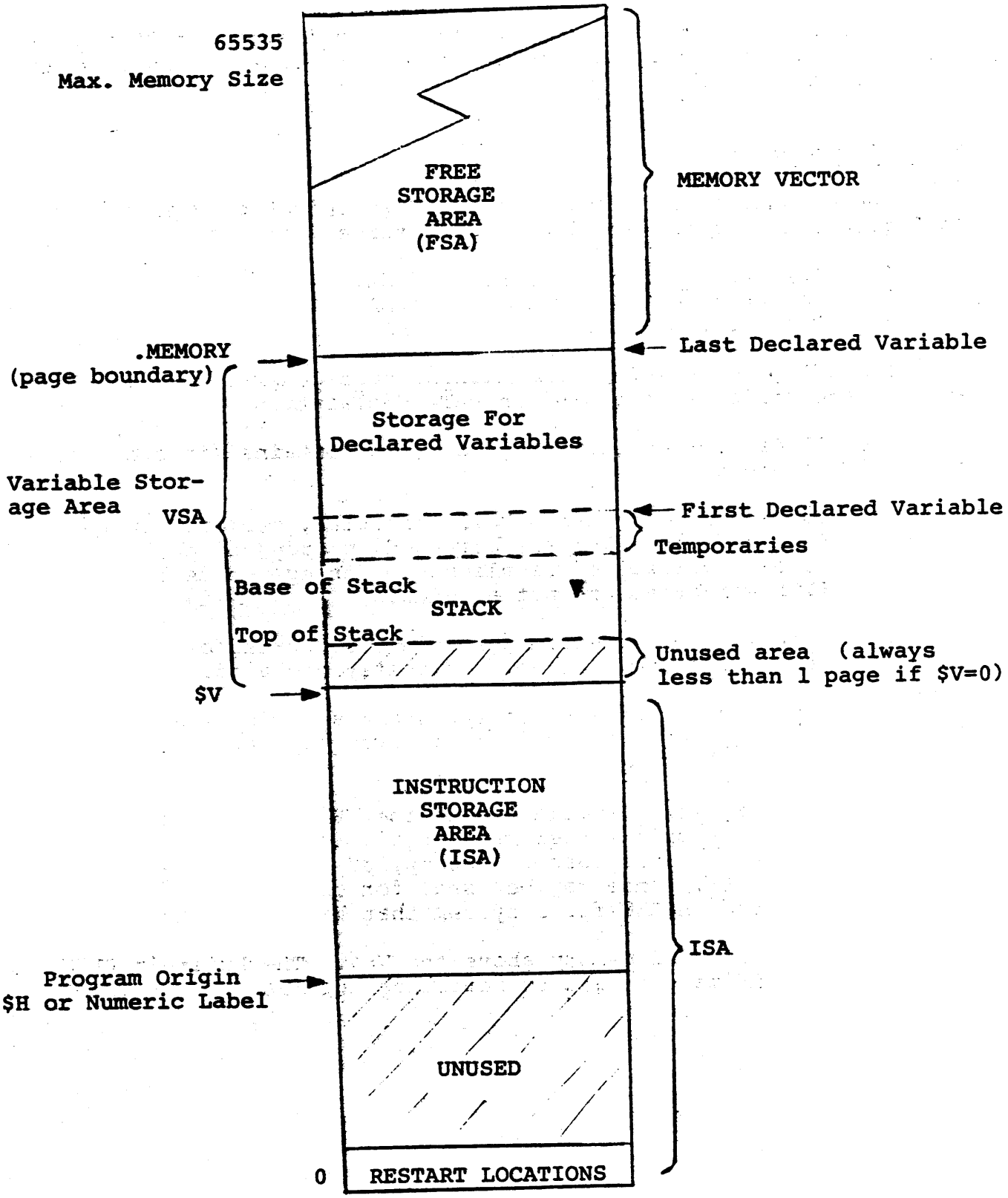

Figure 6-1 Run-time Storage Organization

6.2 Procedure Linkage Conventions

Formal parameters declared in a procedure definition are treated as locally defined variables. That is, each parameter is allocated storage sequentially in memory as if it were a variable local to the procedure. During procedure invocation, actual parameters are evaluated, and the results assigned to the corresponding formal parameters. All parameters are "call by value" in PL/M.

The conventions for passing parameters are as follows:

1. A single BYTE parameter is passed in register C. A single ADDRESS parameter is passed in registers B (high order byte) and C (low order byte).

2. If these are two parameters, the first is passed as described above; the second is passed in registers D (high order byte, if any) and E (low order byte).

3. When there are more than two parameters, the last two are sent as described above, and the remainder are assigned directly, prior to the actual CALL.

CPU registers are also used to hold results returned by procedures which have the BYTE or ADDRESS attribute. In the case of a BYTE procedure, the value returned is in the A register, while an ADDRESS procedure returns the low-order byte in register A, and the high-order byte in register B.

6.3 Use of Assembly Language Subroutines with PL/M

Assembly language subroutines can be incorporated with PL/M programs provided they take account of the PL/M conventions discussed in Section 6.2.

If assembly language subroutines are loaded at addresses S1, S2, . . . Sn (see Figure 6-2), the PL/M program should have interface procedures P1, P2, . . . Pn where each Pi is a procedure containing only the absolute jump:

```
GO TO Si;
```

Each procedure Pi can have up to two parameters of type BYTE or ADDRESS, and can also return a value. If more than two parameters are required or more than one value is to be returned, then ADDRESS variables may be used to 'point to' parameters or results. Each assembly language subroutine Si obtains parameters and returns results according to the conventions presented in Section 6.2.

Suppose, for example, three subroutines are written in assembly language for handling teletype I/O. The subroutine CRLF sends a line-feed-carriage-return, and is at memory location 50. The subroutine TTYOUT writes a single character at the teletype. TTYOUT starts at location 75. The subroutine TTYIN reads one character from the teletype, and is located at address 120. The following PL/M fragment provides appropriate interface procedures:

```
DECLARE CRLFS LITERALLY '50',
        TTYOUTS LITERALLY '75',
        TTYINS LITERALLY '120';

/* INTERFACE FOR CRLF */

CRLF:  PROCEDURE;
         GOTO CRLFS;
         END CRLF;

/* INTERFACE FOR TTYOUT */

TTYOUT: PROCEDURE (CHAR);
         DECLARE CHAR BYTE;
         GOTO TTYOUTS;
         END TTYOUT;

/* INTERFACE FOR TTYIN */

TTYIN: PROCEDURE BYTE;
         GOTO TTYINS;
         END TTYIN;
```

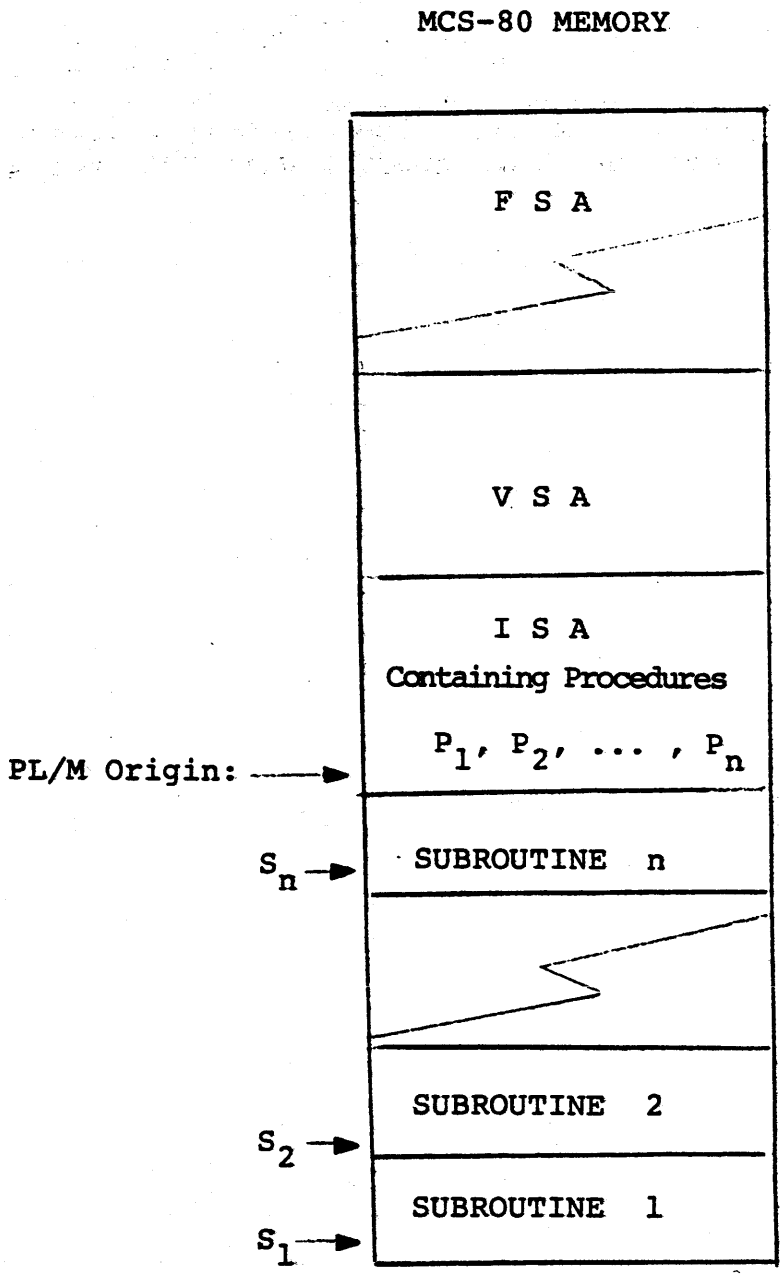

Figure 6-2 Including Assembly Language Subroutines with PL/M Programs.

The user should take care if his assembly language routines make use of the 8080 stack. Firstly, the size of the VSA as determined by the PL/M compiler will take account only of the stack requirements of the PL/M source. Secondly, the assembly language routines, on return, must leave the stack pointer with the same value as it had on entry.

6.4 Stack Manipulation

The use of the 8080 stack is completely automatic in PL/M. It is used, for example, to hold return addresses, temporary results, and system status during interrupt processing.

The number of bytes allocated for the stack is determined during compilation, and assumes no more than one simultaneous activation of any given procedure (including INTERRUPT procedures) at any time during execution.

The 8080 stack pointer register is reset to the address of the base of the stack on the following occasions:

a) Program entry.

b) At numeric labels in the outermost block.

c) Transfers of control to the outermost block from nested inner procedures.

Automatic stack allocation may be bypassed with the $*=n compiler control in pass 2. In this case, no stack area is reserved in the VSA, and the stack pointer is reset to the value n at (a), (b), and (c), above. In this way the programmer may explicitly control the location, and consequently size, of the stack.

The PL/M compiler also provides for stack operation under total control of the programmer. This is accomplished by setting $*=1 during pass 2. In this case, no space is reserved for the stack in the VSA, and no automatic reset of the stack pointer takes place. Its value must be controlled explicitly with the STACKPTR pseudo-variable in PL/M.

6.5 Interrupt Processing

The object code corresponding to a procedure with the INTERRUPT n attribute is such that it may be entered by a transfer of control to location 8n in memory. Location 8n contains a jump to the remainder of the object code of the procedure. Consequently, an interrupt procedure can be invoked by forcing a RST n instruction on the 8080 interrupt port.

Upon execution of the RST n instruction, the current program counter (PC) is pushed on the stack, and control passes via location 8n, to the interrupt procedure. At entry to the interrupt procedure, CPU registers are stacked in the following sequence:

1. (H,L) 2. (D,E) 3. (B,C) 4. (A, Flags).

The interrupt procedure remains active until a corresponding PL/M RETURN statement is encountered, or control passes to the end of the procedure. All stacked registers (except PC) are restored, interrupts enabled, and a RET operation is executed (which restores PC), causing control to return to the point of interruption.

If a PL/M program contains interrupt procedures, locations 0 through 8n + 2 (where n is the highest numbered interrupt procedure) will be reserved for unconditional branches to the procedure bodies. Locations 0, 1, and 2 contain an unconditional jump to the origin of the PL/M program (unless interrupt zero is used).

Note that the 8080 processor starts with interrupts disabled, and disables interrupts when an interrupt is accepted. PL/M object code enables interrupts before returning from an interrupt procedure, and before all program halts.

Appendix A, Error Messages

8080 PL/M COMPILER PASS 1

ERROR MESSAGES

Error Number	Message
1	The symbols printed below have been used in the current block but do not appear in a DECLARE statement; or label appears in a GO TO statement but does not appear in the block.
2	Pass-1 compiler Symbol Table overflow. Too many symbols in the source program. Either reduce the number of variables in the program, or re-compile Pass-1 with a larger Symbol Table.
3	Invalid PL/M statement; the pair of symbols printed below cannot appear together in a valid PL/M statement (this error may have been caused by a previous error in the program).
4	Invalid PL/M statement. The statement is improperly formed - the parse to this point follows (this may have occurred because of a previous program error).
5	Pass-1 Parse Stack overflow. The program statements are nested too deeply. Either simplify the program structure, or re-compile Pass-1 with a larger Parse Stack.
6	Number conversion error. The number either exceeds 65535 or contains digits which conflict with the radix indicator.
7	Pass-1 table overflow. Probable cause is a constant string which is too long. If so, the string should be written as a sequence of shorter strings, separated by commas. Otherwise, re-compile Pass-1 with a larger VARC table.

8 Macro Table overflow. Too many LITERAL declarations. Either reduce the number of LITERAL declarations, or re-compile Pass-I with a larger 'MACROS' table.

9 Invalid constant in INITIAL, DATA, or in-line constant.

Precision of constant exceeds two bytes (may be internal Pass-1 compiler error).

10 Invalid program. Program syntax incorrect for termination of program. May be due to previous errors which occurred within the program.

11 Invalid placement of a declaration within the PL/M program. Declarations may only appear in the outer block or within DO-END groups (not iterative DO's, DO-WHILE's, or DO-CASE's).

12 Improper use of identifier following an END statement. Identifiers can only be used in this way to close a procedure definition.

13 Identifier following an END statement does not match the name of the procedure which it closes.

14 Duplicate formal parameter name in a procedure heading.

15 Identifier following an END statement cannot be found in the program.

16 Duplicate label definition at the same block level.

17 Numeric label exceeds CPU addressing space.

18 Invalid CALL statement. The name following the CALL is not a procedure.

19 Invalid destination in a GO TO. The value must be a label, simple variable, or numeric constant.

20 Macro Table overflow (see error 8 above).

21	Duplicate variable or label definition.
22	Variable which appears in a DATA declaration has been previously declared in this block.
23	Pass-1 Symbol Table overflow (see error 2 above).
24	Invalid use of an identifier as a variable name.
25	Pass-1 Symbol Table overflow (see error 2 above).
26	Improperly formed BASED variable declaration. The form is I BASED J, where I is an identifier not previously declared in this block, and J is an ADDRESS variable.
27	Symbol table overflow in Pass-1. (See error 2 above).
28	Invalid address reference. The DOT operator may only precede simple and subscripted variables in this context.
29	Undeclared variable. The variable must appear in a DECLARE statement before its use.
30	Subscripted variable or procedure CALL references an undeclared identifier. The variable or procedure must be declared before it is used.
31	The identifier is improperly used as a procedure or subscripted variable.
32	Too many subscripts in a subscripted variable reference. PL/M allows only one subscript.
33	Iterative DO index is invalid. In the form 'DO I = E1 to E2' the variable I must be simple (unsubscripted).
34	Attempt to complement a compiler control where the control currently has a value other than 0 or 1.
35	Input file number stack overflow. Re-compile Pass-1 with a larger INSTK table.

36 Too many block levels in the PL/M program. Either simplify your program (30 block levels are currently allowed) or re-compile Pass-1 with a larger Block Table.

37 The number of actual parameters in the calling sequence is greater than the number of formal parameters declared for this procedure.

38 The number of actual parameters in the calling sequence is less than the number of formal parameters declared for this procedure.

39 Invalid interrupt number (must be between 0 and 7).

40 Duplicate interrupt procedure number. A procedure has been previously specified with an identical interrupt attribute.

41 Procedure appears on left-hand side of an assignment.

42 Attempted 'CALL' of a typed procedure.

43 Attempted use of an untyped procedure as a function or a variable.

44 This procedure is untyped and should not return a value.

45 This procedure is typed and should return a value.

46 'RETURN' is invalid outside a procedure definition.

47 Illegal use of a label as an identifier.

8080 PL/M COMPILER PASS 2
ERROR MESSAGES

Error Number	Message
101	Reference to storage locations outside the virtual memory of Pass-2. Re-compile Pass-2 with larger 'MEMORY' array.
102	(Same as 101).
103	Virtual memory overflow. Program is too large to compile with present size of 'MEMORY'. Either shorten program or recompile Pass-2 with a larger virtual memory.
104	(Same as 103).
105	Control used improperly in Pass-2. Attempt to complement a control which has a value other than 0 or 1.
106	Register Allocation Table underflow. May be due to a previous error.
107	Register allocation error. No registers available. May be caused by a previous error, or Pass-2 compiler error.
108	Pass-2 Symbol Table overflow. Reduce number of symbols, or re-compile Pass-2 with larger Symbol Table.
109	Symbol Table overflow (see error 108).
110	Memory allocation error. Too much storage specified in the source program. Reduce source program memory requirements.
111	Inline data format error. May be due to improper record size in Symbol Table file passed to Pass-2.

112 (Same as error 107).

113 Register Allocation Stack overflow. Either simplify the program or increase the size of the Allocation Stacks.

114 Pass-2 compiler error in 'LITADD' -- may be due to a previous error.

115 (Same as 114).

116 (Same as 114).

117 Line width set too narrow for code dump (use $W=n).

118 (Same as 107).

119 (Same as 110).

120 (Same as 110, but may be a Pass-2 compiler error).

121 (Same as 108).

122 Program requires too much program and variable storage.

123 Initialized storage overlaps previously initialized storage.

124 Initialization Table format error. (See error 111).

125 Inline data error. May have been caused by previous error.

126 Built-in function improperly called.

127 Invalid Intermediate Language format. (See error 111).

128 (Same as error 113).

129 Invalid use of built-in function in an assignment.

130 Pass-2 compiler error. Invalid variable precision (not single byte or double byte). May be due to previous error.

131 Label resolution error in Pass-2 (may be compiler error).

132 (Same as 108).

133 (Same as 113).

134 Invalid program transfer.

135 (Same as 134).

136 Error in built-in function call.

137 (Not used).

138 (Same as 107).

139 Error in changing variable to address reference. May be a Pass-2 compiler error, or may be caused by previous error.

140 (Same as 107).

141 Invalid origin. Code has already been generated in the specified locations.

142 A Symbol Table dump has been specified (using the $MEMORY toggle in Pass-1), but no file has been specified to receive the BNPF output (use the $BNPF=n control).

143 Invalid format for the Simulator Symbol Table dump (see error 111).

144 Stack not empty at end of compilation. Possibly caused by previous compilation error.

145 Procedures nested too deeply (HL optimization). Simplify nesting, or re-compile with larger PSTACK.

146 Procedure optimization stack underflow. May be a return in outer block.

147 Pass-2 compiler error in LOADV. Register stack order is invalid. May be due to previous error.

148 Pass-2 compiler error. Attempt to unstack too many values. May be due to previous error.

149 Pass-2 compiler error. Attempt to convert invalid value to address type. May be due to previous error.

150 (Same as 147).

151 Pass-2 compiler error. Unbalanced execution stack at block end. May be due to a previous error.

152 Invalid stack order in APPLY. May be due to previous error.

PASS 1 COMPILER CONTROLS

CONTROL	VALUES	DEFAULT	USE
A	0,1	0	Print syntax analysis trace. (Compiler diagnostic only).
B	0,1	1	Inhibit stack dump after syntax errors. (Compiler diagnostic only).
C			Contains current source line number.
D	Fixed	120	Pass 1 buffer size for output files.
E	0,1	0	Emergency termination when set.
F			Unused
G	0,1	0	Display Intermediate Code. (Compiler diagnostic only).
H			Unused.
*I	1-7	1	File number of Pass 1 input stream.
J	1-7	6	File number for Intermediate Code emitted by Pass 1.
K	Fixed	72	Value assumed by 'W' Toggle for Intermediate Code file.
*L		1	Leftmargin. Specifies first character position processed on each input line. Any leading characters are ignored.
M	0,1	1	Transmit full Symbol Table to Pass 2.
N			Unused
*O	1-7	1	File number for list file.

* Compiler controls identified by an asterisk are the only ones the compiler user should need to use.

Appendix B, Compiler Controls

CONTROL	VALUES	DEFAULT	USE
*P	0,1	1	Echo input if one. Suppress if zero.
Q			Unused
*R	80 Max	72	Rightmargin. Ignore characters R+1, R+2,... on each input record.
S	0,1,2	0	Print Pass 1 Symbol Table. (Compiler diagnostic only).
T	0,1,2	1	0=Batch. 1=Interactive. 2=Interlist. (See Note 1.)
U	1-7	7	Intermediate Symbol Table file number.
V	Fixed	72	Setting of 'W' control for Intermediate Symbol Table file.
W	120 Max	72	Maximum number of characters per record output to the list file. (See notes 2 and 3).
X			Unused
Y	Fixed	1	Output text begins at character position 'Y' of each record. Leading character positions are space-filled. (Applies to all files).
Z			Unused

Note 1. '$T=1' signifies that Pass 1 is operating interactively. This causes output to be double-spaced, with the intention of correctly interleaving it with input. Additionally, if the Pass 1 listing is directed to a device other than the terminal, a summary error report is directed to the terminal at the end of Pass 1.

'$T=0' signifies batch operation. Output is single-spaced but may not be synchronized with the input. The error summary is suppressed.

Note 2. All output from Pass 1 is produced by the subroutine 'WRITEL'. This will write to all files using variable length records in which all trailing space characters are suppressed. Additionally, a leading space, over and above those specified by the controls, is added to each output record as a 'print control character'. Note that this applies also to both intermediate files.

Note 3. Complete echo of the source input requires a width of 94 characters. A setting of the $W control to a value less than this causes one input line to be echoed using two print lines. If the latter portion of the input line is blank this may give the appearance of double spacing.

PASS 2 COMPILER CONTROLS

CONTROL	VALUES	DEFAULT	USE
A	0,1,2	0	Register allocation trace. (Compiler diagnostic only).
B	0,1-7	7	File number for BNPF/Hex output by Pass 2. 0 indicates BNPF/Hex file not to be created.
C			Contains line count from original source file.
D	Fixed	120	Pass 2 buffer size for output files.
E	0,1	0	Emergency termination of Pass 2.
*F	0,1	0	Display decoded memory initialization.
*G	0,1,2	0	0: Off 1: Display cross-reference table of approximate memory address versus source line number. 2: Display intermediate language. (Compiler diagnostic only).
*H		0	Header. Decimal address at which Pass 2 should start allocating space for the generated code. i.e., the start of the program's ISA.
I	1-7	1	File number for Command File input to Pass 2.
J	1-7	6	Intermediate Code file number.
K			Unused

CONTROL	VALUES	DEFAULT	USE
L	Fixed	1	Leftmargin. Specifies first character position processed on each record input from the command file. Leading characters are ignored.
*M	0,1	1	Display Symbol Table.
N	0,1	0	Display emitter trace. (Compiler diagnostic only.)
*O	1-7	1	File number for List File.
P	0,1	0	Echo input. (Compiler diagnostic only).
*Q	0,1	1	0: Object file written in 'BNPF' format. 1: Object file written in Hex format.
R	Fixed	73	Rightmargin. Ignore characters R+1, R+2, ... on each input record. (Applies to Command, Intermediate Code, and Symbol Table Files).
S	0,1,2	0	Display codified Pass 2 Symbol Table. (Compiler diagnostic only).
T	0,1,2	1	See Pass 1 'T' control.
U	1-7	7	File number of Intermediate Symbol Table.
*V		0	Page number of the first page of the VSA. i.e., variable storage, stack, etc. If set to zero: Pass 2 allocates space at the first available page above the ISA.
W	120 Max	72	Maximum number of characters per record output. (Applies to both the list, and BNPF/Hex files. See Note 1).

CONTROLS	VALUES	DEFAULT	USE
X		0	Unused
Y	Fixed	1	Output text begins at character position 'Y' of each record. Leading character positions are spacefilled. (Applies to all output files).
Z	Fixed	2	Value of 'L' control for intermediate files - both Symbol Table and Code.

Note 1. All output from Pass 2 is produced by the subroutine 'WRITEL'. This will write to all files using variable length records in which trailing spaces are suppressed. Additionally, a leading space is added to each output record as a 'print control character'. Note that this applies to the BNPF/Hex file as well as the list file.

GENERAL FILE MAPPINGS

Pass 1

Input		Output	
Control Value	FORTRAN Unit	Control Value	FORTRAN Unit
1	5	1	5
2	2	2	3
3	6	3	7
4	16	4	17
5	9	5	10
6	20	6	22
7	21	7	23

Pass 2

Input		Output	
Control Value	FORTRAN Unit	Control Value	FORTRAN Unit
1	5	1	5
2	2	2	3
3	6	3	7
4	16	4	17
5	9	5	10
6	22	6	20
7	23	7	21

Appendix D, Timesharing File Definitions

GENERAL ELECTRIC FILE DEFINITIONS

Pass 1

Input		Output	
Control Value	File Definition	Control Value	File Definition
1	TTYIN1	1	TTYOUT1
2		2	PTR1
3		3	
4		4	
5		5	
6	FILEIN	6	INTFIL
7		7	SYMFIL

Pass 2

Input		Output	
Control Value	File Definition	Control Value	File Definition
1	TTYIN2	1	TTYOUT2
2		2	
3		3	
4		4	
5		5	
6	INTFIL	6	LOGOUT
7	SYMFIL	7	LOGBIN

TYMSHARE FILE DEFINITIONS

Pass 1

Input		Output	
Control Value	File Definition	Control Value	File Definition
1	Terminal	1	Terminal
2	FOR02.DAT	2	
3	FOR06.DAT	3	FOR07.DAT
4	FOR16.DAT	4	FOR17.DAT
5	FOR09.DAT	5	FOR10.DAT
6	FOR20.DAT	6	FOR22.DAT
7	FOR21.DAT	7	FOR23.DAT

Pass 2

Input		Output	
Control Value	File Definition	Control Value	File Definition
1	Terminal	1	Terminal
2	FOR02.DAT	2	
3	FOR06.DAT	3	FOR07.DAT
4	FOR16.DAT	4	FOR17.DAT
5	FOR09.DAT	5	FOR10.DAT
6	FOR22.DAT	6	FOR20.DAT
7	FOR23.DAT	7	FOR21.DAT

UNITED COMPUTING SYSTEMS FILE DEFINITIONS

Pass 1

Input		Output	
Control Value	File Definition	Control Value	File Definition
1	P801IN	1	P8010UT
2		2	PTR801
3		3	
4		4	
5		5	
6	FILE80	6	INTFIL
7		7	SYMFIL

Pass 2

Input		Output	
Control Value	File Definition	Control Value	File Definition
1	P802IN	1	P8020UT
2		2	
3		3	
4		4	
5		5	
6	INTFIL	6	LOG80
7	SYMFIL	7	LOGB80