PDP16-M

ASSEMBLER DESCRIPTION AND USER'S GUIDE

By

Leonardo A. Uzcátegui

The University of Michigan
Ann Arbor, Michigan

4 September 1973

TABLE OF CONTENTS

1. Introduction

The purpose of this document is to describe the assembler language for the PDP16-M. The reader is assumed to be familiar with the PDP16-M as described in the PDP16-M User's Guide publication of Digital Equipment Corporation (DEC) form number DEC-16-IMUGA-A-D first edition of March 1973.

The PDP16-M assembler is a program written to translate symbolic programs for the PDP16-M into hexadecimal binary programs. These binary programs can be "written" into electrically Programmable Read Only Memories (PROM's) for execution on the PDP16-M. Each PROM contains 256 addressable 8-bit storage locations and is housed in a 24-pin dual-in-line ceramic package. With this in mind, binary programs produced by the assembler can be "written" into the PROM's using the Michigan Intel MCS-8 Programming and Loading (MIM/PL) routines. These routines are described in a different document and information about about them can be obtain from Professors D. E. Atkins and K. B. Irani of the Systems Engineering Laboratory of the University of Michigan.

2. Assembler Statements Format

Each assembler statement may consist of up to four fields: label field, operation code field, operand field, and comments field. If all fields appear in one statement, they must be in that order. The only delimiter between fileds is at least one

blank. Therefore, the format is variable with the exception of the label field which must be in column one if present.

2.1 Label Field

The label field consist of up to eight characters, the first of which must be alphabetic (A-Z), the remaining seven characters may be alphabetic or numeric (0-9). It is used to identify a particular statement with a symbolic name given by the programmer. This field, if present, must begin in column one. Examples of labels are: ADD, SUM2, FINISH, LOOP. The label field is optional.

2.2 Operation Code Field

The operation code field may consist of alphabetic, numeric, or special characters, depending upon which instruction is given. It may consist of all the PDP16-M instructions as described in DEC's publication (with a few exceptions which are indicated in later descriptions) or all the assembler pseudo instructions (see Sec. 5.). Examples are: A=A+1, EXA, A=A.AND.B, TITLE, END, SPACE, IF(A<1>). The operation code field is mandatory.

2.3 Operand Field

The operand field may consist of symbols as described in the label field or special numerics (see Sec. 4.1) or register names (see Sec. 4.2) or strings of characters (see Sec. 4.3).

Some instructions do not require an operand field and therefore it is omitted. Examples are: 0'200', D'920', A'*', START, A, A TITLE.

2.4 Comments Field

The comments field is optional and may follow the operand field or the operation code field. It may consist of any legal characters and indicates descriptive items of information for the programmer. A statement can be all comments by placing an asterisk in column one.

Examples:

```
* THIS IS A COMMENT
      A=A+1      INCREMENT COUNT.
```

Note: As indicated above, the only delimiter between fields is at least one blank. Care must be taken in observing this convention. Henceforth, in this report, delimiters will not be explicitly written. However, it is implied that there is at least one blank between fields.

3. PDP16-M Instruction Types

There are four classes of instructions for the PDP16-M. These are: Register transfer instructions, Unconditional Branch instructions, Conditional Branch instructions, and Subroutine linkage instructions.

3.1 Register Transfer Instructions

The register transfer instructions are used to specify arithmetic, logical, data transfer, I/O, and control operations. The general form of the instructions is:

```
      register=register-operation.
```

Examples of these instructions follow:

```
      A=A+1      (arithmetic group)
```

```
A=A.OR.B      (logical group)
A=TR          (register group)
B=C1          (constant generation group)
A=GPI1        (I/O group)
HALT          (command group)
```

3.2 Unconditional Branch Instruction

The unconditional branch instruction (GOTO) is used to transfer control from one part of the program to another (within the same page. A page is 512 locations). An example of this instruction is:

```
      GOTO ADD
      .
      .
      .
ADD   A=A+B
```

The label 'ADD' must be in the same page as the one in which the GOTO instruction resides.

3.3 Conditional Branch Instruction

The conditional branch instruction (IF) is used to transfer control conditionally from one part of the program to another (in the same page).

There are twenty-one hardwired conditions. That can be tested in the PDP16-M (an additional thirty-six conditions can be obtained with some hardware optional features).

The general form of the conditional branch instruction is:

```
IF(condition) label
```

Examples of this instruction are:

```
      IF(DP) SUB
      .
      .
      .
SUB   A=A-1
```

Here control is transfered to the statement labeled SUB only if the data word is positive. Note that the label must be in the same page.

```
        IF(A<15>) ENDUP
        .
        .
        .
ENDUP   HALT
```

In this case, control is given to the statement labeled ENDUP if bit 15 of register a is set.

3.4 Subroutine Linkage Instructions

The subroutine linkage instructions are used to transfer control to a subroutine (CALL) or to return control from a subroutine to the main program or to another subroutine (EXIT). Here, again, the subroutine to which control is transfered must be within the same page. An example of the use of these instructions is:

```
        .
        .
        CALL ADD
        .
        .
        .
ADD     A=A+B
        B=B+1
        EXIT
```

When control reaches the CALL instruction, it is given to the statement labeled ADD. The instructions A=A+B and B=B+1 are then executed and when the EXIT instruction is performed, control is transfered to the instruction following the CALL instruction.

4. Assembler Pseudo Instructions

Besides the basic PDP16-M, there are several assembler pseudo instructions to enhance the readability of the program listing, and to allow the programmer to equate symbols and predefined registers to more easily remembered names. The assembler pseudo instructions implemented so far are:

```
EQU     - equate symbol
RENAME  - rename predefined register
TITLE   - identify assembly listing
EJECT   - start a new page in the listing
SPACE   - space listing
ORG     - set the location counter
END     - end assembly
```

4.1 Equate Symbol Pseudo Instruction (EQU)

The EQU pseudo instruction is used to define a symbol by assigning to it a value the programmer needs. The general form of this pseudo instruction is:

label EQU operand

The label is any legal symbol for the label field of an statement (see Sec. 2.1). The operand field can be any of three types:

```
O'n' where n is any octal number from 0 to 1777
A'c' where c is any legal ascii character
D'm' where m is any (signed/unsigned) decimal number
     from -1022 to 1023
```

Examples of the use of this pseudo instruction are:

```
RUBOUT EQU O'377'  to refer to O'377' symbolically
BLANK  EQU A' '    to refer to A' ' symbolically
MINUS1 EQU D'-1'   to refer to D'-1' symbolically
```

4.2 Rename Register Pseudo Instruction (RENAME)

The RENAME pseudo instruction is used to allow the name

of predefined registers to become synonymous with names given by the programmer to increase program readability and ease of algorithm implementation. The general form of the RENAME pseudo instruction is:

label RENAME operand-register

The label is any legal symbol for the label field of an statement (see Sec. 2.1). The operand-register can be any of the predefined registers that take part in any of the register transfer instructions (see Sec. 3.1). Examples of the use of this pseudo instruction are:

```
COUNT RENAME A
DATA  RENAME B
```

In this way, COUNT and A are synonymous as well as DATA and B. It is the equivalent to write:

```
      A=A+1
or    COUNT=COUNT+1
or    COUNT=A+1
or    A=COUNT+1
```

Also for A=A+B the programmer could write A=A+DATA or all possible combinations.

4.3 Identify Assembly Listing Pseudo Instruction (TITLE)

The TITLE pseudo instruction is used, as its name indicates, to identify the assembly listing with a title which is printed at the top of every page of the program listing. The general form of the TITLE pseudo instruction is:

TITLE operand

The operand field can be any string of characters that fits in one card image (the remaining spaces after the TITLE instruction is written). This string of characters will be printed at the top of every page of the listing. An example

of the use of this pseudo instruction is:

```
TITLE TELETYPE READER/WRITER ROUTINE
```

As a result, the character string TELETYPE READER/WRITER ROUTINE is printed at the top of every page of the program listing.

4.4 Start New Page Pseudo Instruction (EJECT)

The EJECT pseudo instruction is used to start a new page in the listing of the assembled program. The general form of the EJECT pseudo instruction is:

```
EJECT
```

As aresult, the next line in the listing will be printed at the top of a new page in the program listing. This pseudo instruction is useful for separating routines in the listing.

4.5 Space Listing Pseudo Instruction (SPACE)

The Space pseudo instruction is used to insert one or more blank lines in the assembly listing. The general form of the SPACE pseudo instruction is:

```
SPACE operand
```

The operand may be blank or an unsigned decimal number from 1 to 9. This allows a maximum of nine blank lines to be inserted in the assembly listing. An example:

```
SPACE 3
```

As a result, three blank lines are inserted in the assembly listing at the place where the SPACE pseudo instruction was written.

4.6 Set Location Counter Pseudo Instruction (ORG)

The ORG pseudo instruction is used to set the location

counter to a value defined by the programmer. The general form of the ORG pseudo instruction is:

ORG operand

The operand is defined as in the EQU pseudo instruction described in Sec. 4.1. An example of the use of this pseudo instruction is:

ORG O'200'

This pseudo instruction will set the location counter to a value of 200 octal.

It is important to note htat this pseudo instruction has no relocatable effect whatsoever. Namely, no information is kept in the object module about any of the ORG's pseudo instructions issued. As a result, all the modules produced by the assembler are absolute. (the PDP16-M is not a relocatable machine)

This pseudo instruction is useful for writing page linkage code (seeSec. 4.5.2 in DEC's publication) or if the programmer knows a priori where his/her program will be loaded, he/she can preset the location to to the loading addresss.

4.7 End Assembly Pseudo Instruction (END)

The END pseudo instruction is used to indicate the physical end of the program. It terminates the current assembly. The general form of the END pseudo instruction is:

END

As a result, the assembly is terminated, the object code produced and the listing and cross reference table is printed.

5. Assembler Diagnostics

The assembler continually checks every statement to determine whether it is valid or not. In case an error condition arises, an error message is written after the statement in error. The messages are mostly self-explanatory. A brief description of each of them follows.

*** OPERAND TYPE IS NOT LEGAL.

This message is given whenever the operand type is not either octal, decimal, or ascii.

*** OPERAND DELIMITER IS MISSING.

This message is given whenever one or both primes that delimit the operand is/are missing.

*** OPERAND VALUE IS TOO BIG.

This message is given whenever the value of the operand is greater than 1777 octal or 1023 decimal.

*** OPERAND IS UNDEFINED.

This message is given whenever a label used as an operand has not been yet defined.

*** ILLEGAL BRANCH ADDRESS.

This message is given whenever the branch address is not in the same page in which the branch instruction ocurred.

*** LABEL IS MISSING.

This message is given whenever an instruction which requires a label does not have one. Example: EQU.

*** ILLEGAL USE OF A LABEL.

This message is given whenever an instruction which should

not have a label has one. Example: TITLE, ORG.

*** LABEL IS PREVIOUSLY DEFINED.

This message is given whenever a label is multiply defined. This is the case when two different statements are identified with the same label.

*** ILLEGAL LABEL.

This message is given whenever a label has characters other than alphabetic or numeric, or it does not begin with an alphabetic one.

*** UNDEFINED OPERATION CODE.

This message is given whenever a statement has an operation code which is not either a PDP16-M instruction or an assembler pseudo instruction.

*** UNDECODABLE LINE.

This message is given whenever the assembler cannot interpret an input statement as a legal one. This usually happens when the statement is so wrong that it is almost impossible that it could be a legal statement.

Note: Several additional hardware features may be added to the Basic PDP16-M configuration. Any statement that contains any instruction not available in this basic configuration will be flagged in the assembly listing with an asterisk in column one of the listing. This does not constitute an error and software support is provided to cancel some of these flags as additional hardware is acquired.

6. Object Module and Listing

As indicated in Section 1, the PDP16-M assembler produces object modules which are compatible with the ones produced by the Michigan Intel MCS-8 Assembler (MIM/AL). These object modules are normally produced in the form of a paper tape. This paper tape can then be loaded into the INTEL MCS-8 System using the MIM/PL routines, and then electrically "written" into the PROM'S, from which instructions will be fetched and executed by the program control sequences of the PDP16-M.

The record format in the object module is variable length format with the first byte of each record being a count byte, followed by the data bytes. Each object module is preceded and followed by "leader code"(octal 200) to easily separate modules.

A listing of the program is produced with each assembly. This listing consists of a title page with the model number, system, date and time of the run, followed by the listing of the program which in turn is followed by a cross-reference table for ease in debugging and to identify all the symbols used in the program.

7. Running Environment

The PDP16-M assembler is written to run under the Michigan Terminal System (MTS) and should not be attempted to run under other systems.

The assembler resides in object form in the file SDG6:PDP16M/AS.

Usage: The assembler is invoked by the $RUN command.

Logical I/O units referenced:

SCARDS--The source program to be assembled.
SERCOM--Assembler diagnostics.
SPRINT--Listing and cross-reference table.
SPUNCH--The resulting object module.

Assembler options: The programmer may specify the following options, separated by one comma, in the PAR= field of the $RUN SDG6:PDP16M/AS command. The entries may appear in any order, and, if any are missing, a standard default will be assumed. Following each parameter in the list below is an abbreviated form for the option. The default form of each option is underlined. If any of the entries is not a legal one, the user will be prompted for the legal entry (if in conversational mode).

BATCH	(B)	The assembler processes a stream of assemblies, the last one being terminated with an end-of-file.
NOBATCH	(NB)	After processing one assembly, the assembler returns to the calling program.
DECK	(D)	The object module is produced and written on SPUNCH.
NODECK	(ND)	The object module is not written on SPUNCH.
LINECNT=n	(LC=n)	This specifies the number of lines to be printed between the headings in the source listing. The limits are 1 to 255. The default is 55.
SIZE=n	(S=n)	This specifies the number of virtual K-bytes (1K=1024) to be used for assembling the source program. The limits are 1 to 255. The default is 8.

APPENDIX A

Compatibility with the PAL16 Assembler

This appendix describes the differences between the instruction formats for the PAL16 Assembler and the PDP16-M Assembler as described in this report. The reader is referred to DEC's publication DEC-IMUGA-A-D.

A.1 Comments

The PAL16 Assembler uses the slash (/) to denote a comment follow as in:

```
/ THIS IS AN EXAMPLE OF A COMMENTS LINE
```

OR

```
A=ROM      /FETCH CODE
```

Comments in the PDP16-M Assembler are indicated by an asterisk (*) in column one when the entire line is a comment. For comments on instructions (as in example 2 above), there is no special character to denote them. The two examples given above would read:

```
* THIS IS AN EXAMPLE OF A COMMENTS LINE
```

OR

```
A=ROM      FETCH CODE
```

A.2 Arithmetic Group

The PAL16 Assembler uses the letter 'X' to denote the arithmetic operator 'times' as for example: A=AX2. In the PDP16-M Assembler the asterisk (*) is used instead. for example: A=A*2

A.3 Logical Group

In the PAL16 Assembler, the logical connectives are not delimited in any way from their operands. This would cause considerable problems in "parsing" the operands in the PDP16-M Assembler because of the programmer's ability to rename the operands. For this reason, it was decided to denote the logical operators in the same way FORTRAN denotes them, i.e. by preceding and following them with a period. Examples follow:

```
A=A.NOT.
L=L.NOT.
B=A.XOR.B
A=A.AND.B (in the PAL16, this is written A=AB)
B=A.OR.B
```

A.4 Conditional Jump Group

The IF instruction in the PAL16 Assembler is wtitten by following the word "IF" by at least one blank, then the condition and a comma indicates the end of the condition string. An example is: IF OVF,label. In the PDP16-M Assembler this same instruction is written as: IF(OVF)␣label. Note that there must be at least one blank between the second parenthesis and the label name. However, no blank may appear between the word "IF" and the first parenthesis.