January 1967

The PDPMAP Assembly System

Thomas H. Johnson  
*University of Pennsylvania*

Michael S. Wolfberg  
*University of Pennsylvania*

Follow this and additional works at: [http://repository.upenn.edu/cis_reports](http://repository.upenn.edu/cis_reports)

Recommended Citation


This paper is posted at ScholarlyCommons, [http://repository.upenn.edu/cis_reports/794](http://repository.upenn.edu/cis_reports/794)

For more information, please contact repository@pobox.upenn.edu.
The PDPMAP Assembly System

Abstract
This report describes the PDPMAP Assembly System which is used to assemble symbolic programs written for a Digital Equipment Corporation PDP-8 or DEC-338 with up to 16-K memory locations. The system is implemented at the University of Pennsylvania on an IBM 7040 and DEC PDP-8 connected by a high-speed data channel (IBM 7904 and DEC DM03). The PDPMAP System uses the powerful assembler of a larger computer (IBM 7040 MAP Assembler) to quickly assemble programs for a small computer.

Comments

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/794
INTERIM TECHNICAL REPORT

THE PDPMAP ASSEMBLY SYSTEM

by

Thomas H. Johnson
Michael S. Wolfberg

The following report was prepared under
Contract NOnr 551(40)
for the
Office of Naval Research
and
Rome Air Development Center
and under
Contract AF 49(638)-1421
for the
Air Force Office of Scientific Research

10 October 1967

Moore School Report No. 68-11
Distribution of This Document is Unlimited
TABLE OF CONTENTS

1. Introduction 1
2. Outline of the Assembly System 1
3. Outline of the Assembly Process 3
4. Introduction to the Language 4
5. Absolute vs. Relocatable Assembly 5
6. PDP-8 Instruction Format 6
7. DEC-338 Instruction and Command Format 7
8. Data Generation and Literals 8
9. Origins, Field Setting 9
10. Location Counters 11
11. Location Symbol Definitions 12
12. Absolute Symbol Definitions 13
13. Operation Code Definitions 13
14. Address and Data Transformation 14
15. Memory Reference Instructions and Linking 14
16. Off-Page Linking 15
17. SYMTAB 17
18. Operation of the 7040 18
   18.1 TEST 19
   18.2 PUNCH 21
   18.3 Loading the PDP-8 26
19. Operation of the PDP-8 27
   19.1 PDPMAP-PUNCH 27
   19.2 PDPMAP-LOADER 29
TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20. Postprocessor Output</td>
<td>30</td>
</tr>
<tr>
<td>21. PDPMAP Definitions</td>
<td>32</td>
</tr>
<tr>
<td>21.1 PDP-8 PDPMAP Definitions</td>
<td>32</td>
</tr>
<tr>
<td>21.2 DEC-338 PDPMAP Definitions</td>
<td>33</td>
</tr>
<tr>
<td>22. Defined Operations and Commands</td>
<td>34</td>
</tr>
<tr>
<td>22.1 Initially Defined Operations for PDP-8 PDPMAP</td>
<td>34</td>
</tr>
<tr>
<td>22.2 Initially Defined Operations for DEC-338 PDPMAP</td>
<td>35</td>
</tr>
<tr>
<td>22.3 DEC-330 Control State Commands (microprogrammable)</td>
<td>36</td>
</tr>
<tr>
<td>23. Sample Assembly</td>
<td>37</td>
</tr>
<tr>
<td>23.1 Listing</td>
<td>37</td>
</tr>
<tr>
<td>23.2 Explanation</td>
<td>44</td>
</tr>
</tbody>
</table>

References
THE PDPMAP ASSEMBLY SYSTEM

1. Introduction

This report describes the PDPMAP Assembly System which is used to assemble symbolic programs written for a Digital Equipment Corporation PDP-8 or DEC-338 with up to 16-K memory locations. The system is implemented at the University of Pennsylvania on an IBM 7040 and DEC PDP-8 connected by a high-speed data channel (IBM 7904 and DEC DM03). The PDPMAP System uses the powerful assembler of a larger computer (IBM 7040 MAP Assembler) to quickly assemble programs for a small computer.

2. Outline of the Assembly System

Motivation for construction of the PDPMAP Assembly System stemmed from the problems involved with assembly on the PDP-8 using the standard PAL or MACRO-8 Assemblers. Great amounts of time were used for input/output on the ASR-33 Teletype. As users began writing long programs, they were faced with assembly times measured in hours. Furthermore, the limitation on the number of symbols which could be defined using these assemblers was a severe handicap to programmers.

A PDP-8 assembly system using the 7040 was written at the Moore School by Paul R. Weinberg and Michael S. Wolfberg. This system was satisfactory for a PDP-8 with 4-K memory, but was inadequate for any larger memory or for the DEC-338. Furthermore, in using the assembly system a great amount of programming effort was required to do off-page linking or to assemble complemented variables. By rewriting the system, a more powerful assembly system was produced which provided for the equipment currently in use and for the possible expansion of the equipment, as well as relieving the programmer of some of the problems encountered using the previous assembly system.
The MAP Assembler (IBMAP) is a sophisticated macro assembly program included in the IBSYS operating system for the IBM 7040. IBMAP is used to perform the initial assembly in PDPMAP, and the remaining tasks are carried out by a postprocessor in the 7040. By using the OPD (operation code definition) pseudo-operation of MAP, PDP-8 and DEC-338 operation codes are defined so they are recognizable as MAP instruction mnemonics. Although macros could have been used, the chosen method is significantly faster and produces a more readable assembly listing.

The postprocessor transforms the assembled code and packs it into 12-bit code for the PDP-8. The postprocessor can create off-page links under programmer control and performs other minor functions to enhance programming ease.

The major features of the PDPMAP System are:

1. Fast assembly (a 4-K core load can be assembled on the 7040 in roughly 4 minutes).
2. Similarity with PAL and MACRO-8 and using card input of fixed format.
3. Complete macro capabilities of MAP.
4. Literals (somewhat more limited than the literal facility of MACRO-8).
5. Virtually unlimited symbol table.
6. Automatic creation of off-page links (greater capability than with MACRO-8).
7. Multiple location counters.
8. Line printer listings.
3. Outline of the Assembly Process

The PDPMAP System's assembly process is a multifunctional operation. Different outputs and combinations of outputs can be produced by the Assembly System (see Sections 18 and 19). The assembly process varies mainly in its output. The illustration of process flow discussed below is a general description even though it deals with specific outputs.

A 7040 card deck is submitted as a standard job to be batch processed. The deck consists of

1) IBSYS control cards,
2) the assembly source deck, and
3) a 7040 object program - the PDPMAP Postprocessor - in relocatable binary form.

The MAP Assembler is loaded in the 7040 under directions of IBSYS control cards. The assembly source deck consists of the symbolic program and assembly definitions which must appear at the start of the assembly source deck to condition the MAP Assembler so that it will create a 36-bit "intermediate code" for every assembled line of source code. The intermediate code includes full 15-bit addresses separate from operation codes and indirect bits. Memory reference instructions are tagged so that they can later be easily identified. During the assembly process, a table of used PDP-8 memory consisting of pairs of memory locations, and a table of available linking areas consisting of starting word addresses is maintained.

After the intermediate code has been created (i.e., assembled) by IBMAP, it and the PDPMAP Postprocessor are loaded by the IBSYS loader. Control in the 7040 is given to the PDPMAP Postprocessor.
The PDPMAP Postprocessor scans the intermediate code, word by word, and converts the 36-bit code to the 12-bit code of the PDP-8. If the word being scanned represents a memory reference instruction, the page bit is determined, and it, the indirect bit, and the 7-bit page address are merged with the operation code to form the instruction. If the word being scanned represents an off-page memory reference instruction, a link word containing the off-page address is created on the current page or the initial page of the current field, and a statement of the creation of a link is printed. The off-page memory reference instruction is then changed to a memory reference instruction directly addressing the link word, and the indirect bit is added.

Messages printed by the postprocessor indicate creation of links, illegal off-page references, and insufficient room for creating links.

Prior to this point in the process the PDP-8 is loaded with a short program called PDPMAP-PUNCH in the form of a binary punched paper tape. When PDPMAP-PUNCH is started, it enters a wait loop for the 7040.

Having completed the conversion of intermediate code to 12-bit code, the postprocessor sends the code to the PDPMAP-PUNCH program waiting in the PDP-8. This code is packed so large assemblies can be stored in a small area. PDPMAP-PUNCH performs a checksum of the 12-bit code, dismisses the 7040, and punches the assembled program in the University of Pennsylvania Compressed Binary Loader 8-channel format. The symbol table can be optionally punched in the CBL format for later use with DIE-UP Octal-Symbolic Debugging Program.

4. Introduction to the Language

A set of symbolic cards, PDPMAP Definitions, conditions the MAP Assembler (IBMAP) to accept the PDPMAP Assembly Language. Since IBMAP performs the assembly, PDPMAP source cards are prepared in the same format
as MAP source cards.

A source deck to be assembled begins with a $IBMAP card and a set of thirty cards called PDP-8 PDPMAP Definitions. The DEC-338 PDPMAP Definitions for assembly of DEC-338 programs consist of forty-nine cards. The two sets of PDPMAP Definitions are listed in Sections 21.1 and 21.2.

It is highly recommended that the programmers of this language be rather familiar with the MAP Language, especially with macros. A more thorough facility with MAP can be most helpful in effectively writing programs. The following description of PDPMAP Assembly Language assumes some knowledge of the MAP manual.

5. Absolute vs. Relocatable Assembly

The PDPMAP Assembler assembles only absolute programs for the PDP-8, i.e., the exact loading address is specified at assembly time. The implementation of this assembly system on the 7040, however, uses the MAP Assembler to produce a relocatable binary deck which corresponds to the absolute PDP-8 program. This has made it possible to use the MAP listing in a sensible way - the relocatable location 00000 of the 7040 binary deck corresponds to absolute location 00000 of the PDP-8 program.

In PDPMAP, the symbol "LOC.0" (the last character is a zero) names this relocatable location 00000. In cases where absolute quantities must be specified, such as in a subfield of VFD pseudo-operation, it may be necessary to subtract LOC.0. In cases where relocatable quantities must be specified, particularly in the variable field of a memory reference instruction, it may be necessary to add LOC.0.
6. **PDP-8 Instruction Format**

All PDP-8 operation codes should be punched on cards beginning at column 8. In order to specify indirect addressing for a memory reference instruction, an "*" is included directly following the operation code mnemonic. Memory reference instructions must have an address specified in the variable field. An optional second subfield of the variable field can be used to control off-page linking options. These are discussed in Sect. 16.

Input/Output instructions may stand alone or take one subfield in the variable field. This is useful for CDF and CIF instructions, but the programmer should beware that numbers are normally considered as decimal by the MAP Assembler.

Operate instructions may stand alone or take one subfield in the variable field. Operate microprogramming may be done with two Operate mnemonics by including one in the operation field and one in the variable field. A combination of three or more Operate instructions is specified by including one of them in the operation field and the others in the variable field separated by plus signs (i.e., summed together). When CLA is included with one or more Group 2 Operate instructions, it should be in the variable field and not in the operation field.

There is one exception to the above rules - an excess of 8 must be subtracted from the variable field expression in a combination including SPA, SNA, and SZL.

The PDPMAP Assembler creates a 36-bit intermediate code which for memory reference instructions is:

1) a 1 in bit 2,
2) the indirect flag in bits 12-13,
3) the PDP-8 operation code in bits 15-17,
4) the linking option control in bits 19-20, and
5) the full PDP-8 address in bits 21-35.

The Operate and Input/Output instructions are assembled in the rightmost 12 bits of the 36-bit word. This code appears on the listing printed by the Assembler.

Examples: 

- TAD *+5
- DCA* XYZ
- AND XYZ,2
- KSF
- CDF 8
- CLL
- STL RTR
- SZA HLT
- CLA CLL+CMA
- SMA SZA+CLA
- SPA SNA+SZL-8

7. DEC-338 Instruction and Command Format

In addition to the PDP-8 instructions the DEC-338 PDPMAP Definitions define the twenty Input/Output instructions for the PDP-8 to interact with the DEC-330 Display Processor.

Only Control State Commands are defined for the DEC-330 by the DEC-338 PDPMAP Definitions. They are defined as microprogramming mnemonics to be included in either the operation field or the variable field. Display command microprogramming is specified in the same format used for Operate microinstructions (see Sect. 6).
The JUMP and PJUMP commands are defined as macros which must take a relocatable address and may take a second subfield specifying the extra options which can be combined with these commands.

Examples: SC2

INT 5
LPON SCl+INT+7
EDS SVEC
POP
SK2 INV+COAT
SOPU INTON+BKOF
JUMP XYZ
PJMP XYZ,LPON+SC2

8. Data Generation and Literals

Data words in PDPMAP can be specified by PZE, MZE, DEC, and OCT pseudo-operations and by decimal and octal literals.

If literals are specified in any place in the program either LORG or LITORG pseudo-operations must appear in the PDPMAP source deck.

Examples: PZE 405

OCT 77
PZE -8
MZE 8
TAD =-10
AND =07700
PZE LOOP
MZE LOOP+2
LITORG
9. Origins, Field Setting

At the beginning of an assembly, a default origin of 00000 (LOC.0) is assumed. In order for the PDPMAP Assembly System to be informed of the limits of assembly, the programmer must use one of four special pseudo-operations in order to specify new origins. Each of these pseudo-operations is actually a MAP macro which includes entering the pair of limits into a table of pairs. The // location counter is used to keep track of the pair table. The origin macros depend upon created symbols being ON, which is normally the situation unless a programmer uses the NOCRS pseudo-operation.

Because of the implementation, using BSS or BES pseudo-operations cause the assembly of words (usually zeros) for the PDP-8 in addition to allocating space. It is therefore recommended that origin pseudo-operations be used to allocate blocks of storage.

The following four pseudo-operations are available for specifying origins:

**OCTORG**

This pseudo-operation sets the current assembly location counter to the address specified in the variable field. The address must be an octal number between 0 and 37777.

Examples: OCTORG 10

OCTORG 10000

**ABSORG**

This pseudo-operation sets the current assembly location counter to the address determined by the expression in the variable field. The expression may be any absolute expression whose value is between 0 and 16383.
Examples: ABSORG 8
ABSORG 4096
(Note: These uses of ABSORG are equivalent
to the uses of OCTORG in the examples above.)
ABSORG LOOP-LOC.0
ABSORG SIZE*SIZE+18
(where LOOP is a relocatable symbol and SIZE
is an absolute symbol)

RELORG

This pseudo-operation sets the current assembly location counter to
the address determined by the expression in the variable field. The expression
may be any relocatable expression whose value is between relocatable 0 and
16383.

Examples: RELORG LOOP+15
RELORG LOC.0+SIZE*SIZE+18
RELORG *+10
(where LOOP is a relocatable symbol and
SIZE is an absolute symbol)

PAGE

This pseudo-operation sets the current assembly location counter
to the beginning of a PDP-8 page according to the variable field.

a) A decimal page number between 0 and 127 can be given
which specifies a particular page origin within the 16-K memory.

b) When the variable field is left blank (i.e., no argument is given),
the current assembly location counter is set to the beginning of
the next PDP-8 page. If the location counter is already at the
beginning of a page, this pseudo-operation has no effect. When
no argument is given, a severity zero error message is generated
and should be ignored.

Examples: PAGE 1

PAGE

PAGE 38

FIELD

This pseudo-operation is used to specify the memory field implicitly used by the following pseudo-operations: OCTADR, OCTORG, ABSORG, PAGE (with an argument). It is not necessary to use this pseudo-operation to assemble for higher memory fields; it is included in the PDPMAP System to make it an easy task to take a program previously assembled for one memory field and assemble it for another.

The FIELD pseudo-operation sets the implicit field specification to the set value of the expression in the variable field. This value should be between 0 and 3. In further uses of OCTADR, OCTORG, ABSORG, or PAGE (with an argument), 4096 times the implicit field number is added to the value of the specified expression.

Examples: FIELD 1

FIELD 0

FIELD A

10. Location Counters

The PDPMAP Assembly System uses the // location counter to keep track of limits of assembled words, and therefore the programmer is not permitted to use it. The programmer may, however, define location counters and use them along with the blank location counter if he does so through a special macro. The programmer must not use the USE pseudo-operation.
If a programmer wishes to use location counters in a program, he should include the following macro definition at the beginning of his source deck:

```
USE. MACRO X,Y,Z
    Z RELOQ Y
    ORG Z
    USE X
    Y BSS 0
ENDM USE.
```

This macro can be called in the same way that the USE pseudo-operation is normally called except "USE. PREVIOUS" is not allowed and the specification to use the blank location counter should have "( )" in the variable field.

The **EVEN** pseudo-operation of **MAP** may not be used in **PDPMAP**.

11. **Location Symbol Definitions**

A location symbol must be relocatable in **PDPMAP**. As usual, a location symbol may be defined by including it in the location field of a card containing **PDPMAP** assembly code in the operation and variable fields. The **EQU** pseudo-operation can also be used to define a location symbol as long as the variable field is a relocatable expression.

A special pseudo-operation is available in **PDPMAP** for the definition of a location symbol as an address specified by an octal number.

**OCTADR**

This pseudo-operation takes two arguments - a symbol and an octal number (between 0 and 37777) separated by a comma. Its effect is to define the first argument as a relocatable symbol corresponding to the second argument.
Examples: OCTADR XRL,10010
OCTADR START,200
The programmer should beware not to define location symbols which correspond to any microprogramming mnemonics because these symbols are defined by SET pseudo-operations for use in the variable field of PDP-8 of Operate instructions or DEC-330 commands. Also, LOC.0 is a reserved location symbol naming PDP-8 location 00000, and FLD. is a reserved set symbol used to keep track of the implicit field as set up by the FIELD pseudo-operation.

12. Absolute Symbol Definitions
The EQU, BOOL, and SET pseudo-operations may be used to define absolute symbols. As mentioned in the above section, the programmer should be careful not to define any symbols which correspond to microprogramming mnemonics.

13. Operation Code Definitions
The operation code "END." is used by the PDPMAP Assembly System and should not be redefined by the programmer.

The programmer may define synonyms for any of the six memory reference instruction mnemonics by the OPSYN pseudo-operation.

A special pseudo-operation is provided for the definition of new Input/Output (IOT) and Operate (OPR) instruction mnemonics.

DEFINE
This pseudo-operation takes two arguments - a symbol and an octal number (less than 10000), separated by a comma. Its effect is to define the first argument as an operation code whose value is the second argument. The operation code may either be used alone or with one subfield in the variable field. If the second argument is greater than or equal to 7000
(octal), the first argument is also defined with a SET value corresponding to the low-order eight bits of the second argument. This makes it possible to include the symbol in the variable field of an Operate instruction for microprogramming purposes.

Examples: DEFINE STA,7240
DEFINE LCC,6703

14. Address and Data Transformation

The PDPMAP Postprocessor transforms the 36-bit intermediate code assembled by MAP into 12-bit PDP-8 code. The absolute value of absolute expressions to be assembled must be less than 4096 (10000 octal) or greater than 28672 (70000 octal), otherwise the quantity is assumed to be an address which must be unrelocated (i.e., the postprocessor will subtract LOC.0 from the address). For all absolute expressions the low order 12 bits of the address are used. This makes it possible to specify expressions with a negative value in the address of a PZE.

If the sign bit of the 36-bit intermediate code is a ONE the two's complement of previously computed 12-bit quantity is taken. This gives the programmer the power to specify negative numbers in literals or in OCT or DEC pseudo-operations. It also permits the programmer to specify the complement of a relocatable address by using MZE. Using PZE with a negative address does not work for this purpose.

15. Memory Reference Instructions and Linking

The PDPMAP Postprocessor recognizes a memory reference instruction when bit 2 of the 36-bit word is a ONE. If the instruction addresses a location on the same page or initial page of the current field, a 12-bit PDP-8 word is computed. If the memory reference instruction does not have an indirect
flag and if the address cannot be directly addressed, the postprocessor will attempt to create a link. The linking algorithm is described below.

If the memory reference instruction has an indirect flag and if the address cannot be directly addressed in the PDP-8, an error diagnostic is listed and the 12-bit quantity is computed as if the address were on the same page. If the memory reference instruction has an indirect flag and if the address is on a different field than the current location, an error diagnostic is listed and the 12-bit quantity is computed as if the address were on the same field using the previous algorithm for indirect addressing.

16. Off-Page Linking

The programmer using PDPMAP does not have to perform the bookkeeping often needed to properly address locations in the PDP-8. He can let the postprocessor create the necessary links to referenced locations when necessary. The programmer can control the postprocessor's linking through a special pseudo-operation and some options to alter the normal linking algorithm.

LINKS

This pseudo-operation is used to indicate where on a page links can be used or created. The use of this pseudo-operation does not alter the current location counter; it indicates that the locations between its occurrence and the end of the page or the next origin in the source deck, are considered available linking area. If the next origin is beyond the end of the page the area stops at the end of the page. If the next origin is less than the current location the link area goes to the end of the page.

The LINKS pseudo-operation can be used any number of times.

Any unused location in the available linking area is considered a possible location for the generation of a link word. Any used location in
the available linking area which does not contain a memory reference instruction is considered an existing usable link word according to the 12-bit quantity assembled there.

In the absence of any programmer alteration, links are created for memory reference instructions which reference locations not on the current page or the initial page. Exceptions occur with references to locations on other fields. For a direct JMP or JMS instruction referencing a location on a field other than the current one, the postprocessor assumes a CIF instruction has been given previously and will assemble the instruction as if the referenced location were in the current field. For a direct AND, TAD, ISZ or DCA instruction referencing a location on a field other than the current one, the postprocessor assumes a CDF instruction has been given previously and assembles the instruction with a link no matter what location is being referenced.

The linking algorithm used by the postprocessor in the absence of any programmer alteration options is:

1. Attempt to use an existing linking word on current page.
2. Attempt to use an existing linking word on the initial page of the current field.
3. Attempt to create a linking word on the current page.
4. Attempt to create a linking word on the initial page of the current field.
5. If all attempts fail, the 12-bit quantity is computed as if the address were on the same page except cross-field references to an initial page are computed as if the address were on the initial page of the current field.
6. If all attempts fail (1-4) an error diagnostic is listed.
The PDPMAP Postprocessor operates on the coding in order, from location 0 through the end of memory. Links are created in the order they are processed and not the order in which the program was assembled. In a similar fashion, the linking areas specified by the LINKS pseudo-operation are filled from low address to high address. If there is more than one linking area on a page the one with the lower address is filled first.

It is important to note that by proper use of the origin pseudo-operations the programmer can overlay part of his coding. This does not overlay the LINKS pseudo-operation. The linking area specified early in an assembly will remain a linking area even though new code may overlay it.

Extra control over the linking algorithm is provided by using the tag (second subfield of the variable field). The following three options are provided:

<table>
<thead>
<tr>
<th>TAG</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Do not link - if a link is needed, assemble as if the address is on the current page except cross-field references to an initial page, then assemble as if the address were on the initial page of the current field (Step 5 only)</td>
</tr>
<tr>
<td>2</td>
<td>Consider linking on the initial page only (Steps 1, 3, 5, and 6)</td>
</tr>
<tr>
<td>3</td>
<td>Consider linking on current page only (Steps 2, 4, 5, and 6).</td>
</tr>
</tbody>
</table>

Any tags greater than 3 are ignored. When no linking is needed, the tag is ignored.

17. SYMTAB

It is often desirable to obtain a symbol table paper tape as an auxiliary output from the PDPMAP assembly process. This tape can be later read into DDT-UP so the user may debug his program using the same symbols in which his program was written. Such a symbol table includes only location symbols and those defined by EQU, BOOL, and OCTADR pseudo-operations. No created symbols or operation codes are included in the symbol table.
The creation of the symbol table is performed on the 7040 by the
SYMTAB program, and the results are directly sent to the PDP-8 to be punched
as a CBL binary paper tape by the PDPMAP-PUNCH program. SYMTAB reads a
Debug Dictionary produced by IBMAP as input data to get the information necessary
to output a DDT-UP symbol table. The Debug Dictionary is normally obtained
in a 7040 IBSYS/IBJOBS run to facilitate the use of an execution-time debugging
package. It is punched by IBMAP when a binary deck is punched and when the
DD option is included on the $IBMAP card.

In the PDPMAP System, a user may obtain his symbol table tape when
the assembly is performed or it may be obtained by a separate run. The various
ways of setting up the 7040 run are individually discussed below. Operation
of the PDPMAP-PUNCH program used to punch a symbol table on the PDP-8 is
described in Section 19.1.

The SYMTAB program has the option, according to which entry point is
used, of punching any binary deck it reads (except for the Debug Dictionary).
This is useful in the type of run described in Section 18.2.

18. Operation of the 7040

The 7040 jobs which are used in the PDPMAP System execute under
standard batch IBSYS. This section describes the ways in which these jobs
are prepared. There are three basic options of the PDPMAP Postprocessor:
TEST, PUNCH, LOAD. The option is indicated by punching one of the three in
column 16 of the PDPMAP Postprocessor's $ENTRY card. There is also the facility
to obtain a DDT-UP symbol table, and there is the facility to punch a binary
relocatable card deck.

The most common variations of all these options are described below,
including listings of control cards to be used. In the listings of control
cards, the symbol "NAME" is the arbitrarily chosen name of the user's PDP-8
program.
In any of the variations which include the PDP-8, either machine may begin execution before the other.

18.1 TEST

The Testing processes take place entirely in the 7040 with no interaction with the PDP-8. Testing is useful for checking the assembly or determining if linking can be performed when the PDP-8 is not available or when using a 7040 with no PDP-8.

Two levels of testing are available.

Assemble Only

The first level of testing only uses the MAP assembler to assemble the user's symbolic deck. This type of run catches undefined and multiply defined symbols, illegal operation codes, problems in macro expansions, etc. This run is set up as follows:

$JOB ...
$IBJOB
$IBMAP NAME

PDFMAP definitions

User's program

END

$IBSYS

Assemble and Test

A second level of testing may be performed by running the PDPMAP Postprocessor on the 7040 to create links and check for illegal addressing. This level of testing will be performed as long as a binary deck would normally be punched by MAP. Low severity errors such as undefined and multiply defined symbols will not stop execution of the PDPMAP Postprocessor. The occurrence
of an illegal operation code, however, will block the second level of testing, but the first level of testing is still performed. This use of the PDPMAP Postprocessor on the 7040 does everything the program does when the 'PUNCH' option is used except results are not sent to the PDP-8 (Section 18.2). Therefore, this testing could be performed on another 7040.

The run is set up as follows:

```
$JOB ...
$RESTORE
$CLOSE   S.SU15,REWIND
$SWITCH S.SPP1,S.SU15
$IBJOB '  DECK
$IBMAP NAME

    PDPMAP
    definitions

    User's
    program

    END

$IBSYS
$SWITCH S.SPP1,S.SU15
$CLOSE S.SU15,MARK,REWIND
$IBJOB PDPMAP NOMUX
$IBLDR PDPMAP

    Postprocessor
    binary deck

$ENDPDPMAP
$EDIT U15
$IBLDR NAME
$EDIT IN
$ENTRY TEST
$IBSYS
$RESTORE
$IBSYS
```

Note that the name of the user's program on the $IBMAP card must correspond to the name on the last $IBLDR card.
Assemble, Get Deck, Test

Since the 7040 connected to the PDP-8 uses an on-line card reader and line printer, it may be desirable to run a long assembly on another 7040 and just punch a binary deck. In addition to punching, the user may as well test the results by executing the PDPMAP Postprocessor in the test mode. Such a run is set up as follows:

$JOB ...
$IBJOB PDPMAP NOMAP,DECK
$IBLDR PDPMAP
 Postprocessor
 binary deck
$IKEND PDPMAP
$IBM MAP NAME
 PDPMAP
 definitions
 User's
 program
 END
$ENTRY TEST
$IBSYS

18.2 PUNCH

If the user wishes to obtain a CBL tape of his program, the PDPMAP Postprocessor must be run on the 7040 and the PDP-8 must be loaded and started with the PDPMAP-PRINT program. The operation of the PDPMAP-PRINT program on the PDP-8 is described in Section 19.1. The `PRINT` option is the standard PDPMAP option with the variation of runs listed below.

Assemble, Get Tape

The recommended standard PDPMAP run includes both the assembly of the symbolic deck and the execution of the PDPMAP Postprocessor. No binary cards are punched by this type of run. The run is set up as follows:
Note that the only difference between the above set of cards and those in Section 18.1 is with the $ENTRY card. Alternatively, the $ENTRY could be included with no name. In this case the assumed option is 'PUNCH'.

Assemble, Get Tape - Method No. 2

If the user is very sure of his program's correctly assembling, he may wish to use a set of control cards simpler than those listed above to achieve the same effect. The run is set up as follows:
Assemble, Get Deck, Get Tape

If the user wishes to assemble, obtain a binary deck, and punch a CBL tape all in one run, then he must have an assembly with no errors of severity greater than one. Undefined or multiply defined symbols will block the execution of the PDPMAP Postprocessor although a binary deck will still be punched. It could subsequently be included in a run of the type described in the next subsection to create the CBL tape.

A run of this type is set up as follows:

```
$JOB ...
$IBJOB PDPMAP NOMAP
$IBLDR PDPMAP
    Postprocessor
    binary deck
$DKEND PDPMAP
$IBMAP NAME
    PDPMAP
definitions
    User's
    program
    END
$ENTRY   PUNCH
$IBSYS
```
Get Tape

If the user has already obtained a binary deck (such as in the previous subsection), then he may submit a 7040 run which will be very quick. This run will result in a CBL tape in addition to the output listing generated by the PDPMAP Postprocessor on the 7040.

The run is set up as follows:

$JOB ...
$IBJOB PDPMAP NOMAP,NOSOURCE
$IBLDR PDPMAP
   Postprocessor
   binary deck
$DKEND PDPMAP
$IBLDR NAME
   binary deck
$DKEND NAME
$ENTRY PUNCH
$IBSYS

Assemble, (Get Binary Deck), Get CBL Tape, Get Symbol Table

This variation is the recommended standard for short or moderate length assemblies. When running this variation, the PDP-8 should be set up to automatically restart the PDPMAP-PUNCH program (see Section 19.1).

The user has the option of obtaining a binary deck (without the Debug Dictionary). The $ENTRY card for SYMTAB is used to indicate whether a binary deck is punched. The user may specify either 'DECK' or 'NODECK' starting in column 16, or if neither is there the assumed option is NODECK.

The run is set up as follows:
Note that the name of the user's program on the $IBMAP card must correspond to the name on the last $IBLDR card.
Symbol Table Only

If the user previously assembled his program and obtained a binary deck including a Debug Dictionary, then he may wish to perform this type of run. Such a binary deck could have been punched as described in Sections 18.1 or 18.2 except, in addition, the options on the $IBMAP card should include 'DD'.

The only data cards necessary for this job are the Debug Dictionary cards from the $DDICT through the last binary card before $TEXT. The user may, however, include the entire binary deck as data, and the extra cards will be ignored.

SYMTAB will read any number of Debug Dictionaries provided as input so that symbol tables of separate assemblies could be combined at this level of operation since one DDT-UP symbol table tape results from this process.

The run is set up as follows:

$JOB ...
$IBJOB SYMTAB NOSOURCE,NOMAP
$IBLDR SYMTAB

binary deck

$DKEND SYMTAB
$ENTRY
$DDICT NAME

binary cards
Debug Dictionary

$IBSYS

18.3 Loading the PDP-8

A program which has been assembled by the PDPMAP Assembler may be quickly loaded into the PDP-8 directly from the 7040. The binary deck of the assembled program is used, and the LOAD option of the PDPMAP Postprocessor is used (by specifying 'LOAD' on the $ENTRY card). This method of loading always
loads PDP-8 locations 0000 through 7575, and locations where nothing was assembled are cleared to 0000.

Operation of the PDPMAP-LOADER program used on the PDP-8 is described in Section 19.2.

This run is set up as follows:

$JOB ...
$IBJOB PDPMAP NOSOURCE,NOMAP
$IBLDR PDPMAP
   Postprocessor
   binary deck
$DKEND PDPMAP
$IBLDR NAME
   binary deck
$DKEND NAME
$ENTRY LOAD
$IBSYS

19. Operation of the PDP-8

This section describes the operations of the PDP-8 in the PDPMAP Assembly System. There are two PDP-8 programs involved. The PDPMAP-PUNCH program is used to receive data from the 7040 and punch a CBL binary tape (either user's program or symbol table). The PDPMAP-LOADER program is used to receive data from the 7040 such that the PDP-8 is quickly loaded, and there is an option to immediately begin execution.

19.1 PDPMAP-PUNCH

This program is loaded into locations 0000 through 0377 of the PDP-8 by the CBL Loader. The operation of this program follows:

a) START the PDP-8 at location 0000. The program goes into a loop to wait for the 7040 to send the data. If the 7040 was already waiting, transmission occurs immediately.
b) If no successful transmission was made, the program remains in the wait loop for the 7040. If transmission was completed, a checksum of the data is computed. If the checksum is correct, the program proceeds to examine the SWITCH REGISTER setting. If the checksum is incorrect, the transmission is attempted nine more times. Should all ten attempts fail, the PDPMAP Postprocessor prints an error message and returns control to IBSYS.

c) If bit 0 of the SWITCH REGISTER is a ONE, the PDP-8 halts after successful transmission. Depressing CONTINUE causes the program to proceed to the next step.

d) The CBL tape is punched beginning with a few inches of leader and ending with a few inches of blank trailer. Be sure the Teletype punch is turned on and the Teletype is on-line.

e) After the tape has been punched, the SWITCH REGISTER is again examined by the program. If bit 1 of the SWITCH REGISTER is a ZERO, the PDP-8 halts. Depressing CONTINUE causes the program to proceed to the next step.

f) If bit 1 of the SWITCH REGISTER is not ZERO, the program is restarted at location 0000 for another job. If the 7040 had already been waiting for the PDP-8, transmission will occur immediately.

The first option of halting (step c) is useful when only one PDPMAP punching is desired. The PDP-8 can be set up to wait (unattended) with its Teletype turned off until the 7040 PDPMAP job is run. If successful transmission occurs, the PDP-8 will halt. At a later time, the user may return to find the job completed. He then may turn on the Teletype to 'LINE', turn on the Teletype punch, and depress CONTINUE to commence punching.
If there is a transmission error, and the user decides to take his chances of punching a possibly invalid tape, he may force the PDPMAP-PUNCH program to begin operation at step d by STARTing at location 0001. This option can also be used to punch more than one copy of the CBL tape or to restart punching if the user forgot to turn on the Teletype punch.

19.2 PDPMAP-LOADER

This program is loaded into locations 7600 through 7667 of the PDP-8 by the CBL Loader. The operation of this program follows:

a) LOAD ADDRESS with 7600.

b) For immediate execution of the program being loaded, set the SWITCH REGISTER to the appropriate starting address. If 7600 is left in the SWITCH REGISTER, the PDP-8 will halt after successful loading.

c) START the PDP-8. The PDPMAP-LOADER goes into a loop to wait for the 7040 to send the data. While it is waiting, the accumulator lights indicate the contents of the SWITCH REGISTER when the program was started. If the 7040 was already waiting, transmission occurs immediately.

d) A checksum of the data is computed, and if transmission was successful, the program acts according to the way it was started in steps b and c. If the halting option was used, depressing CONTINUE causes the PDP-8 to jump to location 7777, which is the starting address of the CBL Loader.

e) If no successful transmission was made, the program remains in the wait loop for the 7040.

The PDPMAP-LOADER is self-initializing and self-preserving. It may therefore be used any number of times.
20. Postprocessor Output

When the PDPMAP Postprocessor is operated in 'TEST' or 'PUNCH' mode a listing is printed indicating the formation of links and errors that occur during postprocessing. In Section 23 there is a sample listing of a PDPMAP program with the postprocessor output at the very end.

The column at the far left of the output labeled PAGE STARTING ADDRESS contains the octal starting addresses of pages assembled by PDPMAP. This address is printed immediately following the other information about the page of assembled code.

The next three columns are utilized when links are created. The column labeled LINKED INSTRFUCTION ADDRESS contains the location of an instruction which was linked through the address found in the next column labeled LINK WORD ADDRESS. The value stored in the link word is printed in the next column under LINK WORD VALUE. If this number is in parentheses then the value existed previously.

The last three columns are printed on the same row as the page starting address and they summarize the number of linking operations done on that page. The first two columns, below LINKS CREATED, indicate the number of links created on the current page and the initial page respectively. The last column indicates the number of link words on the current page that are not used by the assembled program.

At the end of every field where linking could be done through its initial page the final status of the initial page is printed out listing the total number of links created on the initial page and the number of link words still available.
When the PDPMAP Postprocessor operates in 'LOAD' mode, only error diagnostics are printed.

**Error Diagnostics**

Error diagnostics at assembly time originate from the MAP assembler.

Errors detected in the PDPMAP Postprocessor are concerned with incorrect addressing or communication with the PDP-8.

**UNABLE TO LINK** LOC. = \{a\} TO LOC. = \{b\}

There are an insufficient number of link words to link the instruction in location "a" to the address "b". The cause is neglecting to use the LINKS pseudo-operations or running out of link words.

**LOCATION \{a\} IS INCORRECTLY Addressed TO \{b\}**

A memory reference instruction with an indirect flag cannot directly address the referenced location. The location is off the page or off the field.

**PROGRAM EXCEEDS PUNCH BUFFER** \{a\} IS THE LAST LOC. PUNCHED

Only locations from the beginning of core to "a" could be punched at the PDP-8.

**COMMUNICATION ERROR**

The data channel between the 7040 and the PDP-8 has malfunctioned.

This error should be brought to the attention of the proper personnel for correction.
21. PDPMAP Definitions
21.1 PDP-8 PDPMAP Definitions

UNLIST PDP-8 PDPMAP DEFINITIONS 7/1/67 (30 CARDS) 00000001
A MACRO H,J,K,L,M,N,P THESE 30 CARDS DEFINE THE BASIC PDP-8 00000002
H J INSTRUCTION SET PLUS THE FOLLOWING ... 00000003
K L BY MEMORY FIELD SETTING - 'FIELD' 00000004
M N M.S. NEW OPR OR IOT DEF'N. - 'DEFINE' 00000005
IRP P WOLFBERG OCTAL ADDRESS DEF'N. - 'OCTADR' 00000006
P ORIGINS - 'ABSORG', 'RELORG', 'OCTORG', 'PAGE' 00000007
ENDM A REGION FOR OFF-PAGE LINKS - 'LINKS' 00000008
LOC.O A END.(OPSYN END)B(MACRO X)(I)(IRP X)(I)(DEFINE X)ENDM(ENTRY LOC.O) 00000009
A END(MACRO X)(X)(BSS O)(I)(USE //)((PZE -1,X)(PMC ON)ENDM,ENDM) 00000010
A LINKS(MACRO X)(X)(BSS O)(I)(USE //)((MZE X)(USE PREVIOUS)ENDM) 00000011
A PAGE(MACRO X,Y)(Y)(EQU */-LOC.O-FLD.+127)()ENDH X 00000012
ETC ((IFF 1'X')(DUP 2,0)(ABSORG Y/128*128)IFF(ABSORG 128*X)ENDM) 00000013
A OCTORG(MACRO X,Y)(Y)(REAL (X)(ABSORG Y)ENDM(BEGIN ++)++16384)) 00000014
A DEFINE(MACRO X,Y)(X)(OPD 10000*Y*000000,*,1,2))ENDM() 00000015
A ABSORG(MACRO X,Y,Z)(X)(BOSS O)Z(ORG X+LOC.O-FLD.) 00000016
ETC ((USE //)((PZE Z,Z,Y)(USE PREVIOUS)ENDM(B ((AND,O)(TAD,1)))) 00000017
A RELORG(MACRO X)(X)(ABSORG X-LOC.O-FLD.)()ENDM(B ((ISZ,2))) 00000018
A (I(B ((DCA,3),(JMS,4),(JMP,5))FIELD(MACRO X)FLD.(SET X*4096)ENDM)00000019
A FLD.(SET 0)DEFINE(MACRO X,Y)(X)(OPD Y,*,1,((IFF Y/7000))) 00000020
X SET Y-Y/10*2-Y/100*16-Y/1000*640-Y/7400*56 00000021
ENDM DEFINE
B ((OOR,7000)(NDP,7000)(CLA,7200)(CLL,7100)(CMR,7040)(CML,7020)) 00000022
B ((RAR,7010)(RAL,7004)(RTR,7012)(RTL,7006)(IAC,7001)(SMA,7500)) 00000023
B ((SZA,7440)(SPA,7510)(SNA,7450)(SNL,7420)(SZL,7430)(OSR,7404)) 00000024
B ((HLT,7420)(CLA,7041)(LAS,7604)(STL,7210)(GLK,7204)(SKP,7410)) 00000025
B ((IOT,6000)(ION,6001)(IOF,6002)(KSF,6031)(KCC,6032)(KRS,6034)) 00000026
B ((KRB,6036)(TSF,6041)(TCC,6042)(TPC,6041)(TLS,6046)(CDF,6201)) 00000027
B ((CIF,6202)(RDF,6214)(RIF,6224)(RMF,6244)(RIB,6234)) 00000028
21.2 DEC-338 PDPMAP Definitions

UNLIST DEC-338 PDPMAP DEFINITIONS 7/1/67 (49 CARDS) 00000001
A MACRO H,J,K,L,M,N,P THESE 49 CARDS DEFINE THE BASIC DEC-338 00000002
H J INSTRUCTION SET PLUS THE FOLLOWING ...
K L BY MEMORY FIELD SETTING - 'FIELD'
M N M.S. NEW OPR OR IOT DEF*N. - 'DEFINE'
IRP P WOLFBERG OCTAL ADDRESS DEF*N. - 'OCTADR'
P ORIGINS - 'ABSORG', 'RELORG', 'OCTORG', 'PAGE' 00000007
ENDM A REGION FOR OFF-PAGE LINKS - 'LINKS'

...OC 0 A END. (OPSYN END) (MACRO X)((IRP X)((DEFINE X) ENDM(ENTRY LOC.0))) 00000009
A END(MACRO X)(BSS 0)((USE //)((PZE -1,x)(PMG ON)END,ENDM)) 00000010
A LINKS(MACRO X)(BSS 0)((USE //)((MZE X)(USE PREVIOUS)ENDM)) 00000011
A PAGE(MACRO X,Y)(EQU -=LOC.0-FLD.+127))((IFT X)
ETC ((IFF 1*X)(DUP 2,0)(ABSORY/128*128)(IFF(ABSORG 128*X))ENDM) 00000013
A DEFINE(MACRO X,Y)(X(OPD 10000*Y'00000,*1,2))((ENDM))
A ABSORG(MACRO X,Y,Z)(BSS 0)Z(ORG X+LOC*0+FLD)) 00000016
ETC ((USE //)((PZE Z,Y)(USE PREVIOUS))ENDM(B ((AND O)TAD1))) 00000017
A RELORG(MACRO X)(ABSORY-X-LOC.0-FLD.))ENDM(B ((ISZ 2))) 00000018
A () (()) DEFINE(MACRO X,Y)(X(OPD Y++1))X(SET Y-Y/10*2-Y/100*16) 00000021
ENDM DEFINE

B ((LPFD,40)(LPON,60)(SC1,400)(POINT,1100)(INCR,1110)(VEC,1120)) 00000023
B ((SC2,500)(SC4,600)(SC8,700)(VECON,1130)(SVEC,1140)(CHAR,1150)) 00000024
B ((INT,10)(EDS,1001)(CCS,1002)(CSB,1004)(PSNLS,304)(PNM,3010)) 00000025
B ((GRAPH,1160)(CLDF,1200)(STOP,1400)(PEDS,3001)(PNI,3002)) 00000026
B ((SKIP,6240)(SNSZ,6220)(SPB1,6210)(SPB2,6204)(SCUP,6340)) 00000027
B ((SCDN,6360)(INTUP,6310)(INTDN,6314)(BKON,6302)(BKOF,6301)) 00000028
A JUMP(MACRO A,B,C,D)(EQU A-LOC.0)DISET B)
ETC ((PMC(PZE 1024+C/4096+D)(PZE A)PMC,ENDM(B ((CLPS1,6201)))) 00000030
A PJMP(MACRO A,B,C,D)(EQU A-LOC.0)DISET B)
ETC ((PMC(PZE 1032+C/4096+D)(PZE A)PMC,ENDM(B ((SLPS2,6202)))) 00000031
A FLD.(SET 0)DEFINE(MACRO X,Y)(X(OPD Y++,1)X(SET Y-Y/10*2-Y/100*16)) 00000032
B ((OPR,7000)(NOP,7000)(CLA,7200)(CLL,7100)(CMA,7040)(CML,7020)) 00000036
B ((RAR,7010)(RAL,7004)(RTR,7012)(RTL,7006)(IAC,7001)(SMA,7500)) 00000037
B ((SHA,7440)(SPA,7510)(SNA,7450)(SML,7420)(SZL,7430)(USR,7404)) 00000038
B ((HLT,7402)(CIA,7041)(LAS,7604)(STL,7120)(GLK,7204)(SKP,7410)) 00000039
B ((IOD,6000)(ION,6001)(IOF,6002)(KSF,6031)(KCG,6032)(KRS,6034)) 00000040
B ((KRB,6036)(TSF,6042)(TCF,6042)(TBC,6044)(TLS,6046)(CDF,6201)) 00000041
B ((CIF,6202)(RDF,6214)(RIF,6224)(RDF,6244)(RIF,6234)) 00000042
B ((SLPL,6132)(SPES,6151)(SPFS,6152)(SPSF,6171)(SPML,6172)) 00000043
B ((POP,3000)(SK1,4000)(SK2,5000)(SK3,6000)(SK4,6100)) 00000044
B ((RPDP,6051)(RXP,6052)(RYP,6054)(RDP,6054)(RAC,6051)(RS1,6062)) 00000045
B ((RS2,6064)(RPB,6071)(SPDP,6135)(SIC,6145)(LBF,6155)) 00000046
B ((INIT,6165)(RES1,6174)(RES2,6164)(CDF,6161)(STPD,6154)) 00000047
A INV(SET 256)CLAT(SET 128)COAT(SET 64)1)
A OCTADR(MACRO X,Y,Z)(BOOL Y)(X(EQU LOC.0+Z+FLD.))(ENDM,LIST) END 00000049
### 22. Defined Operations and Commands

#### 22.1 Initially Defined Operations for PDP-8 PDPMAP

<table>
<thead>
<tr>
<th>Memory-Reference Instructions</th>
<th>Group 1 Operate Instructions (microprogrammable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND 0000</td>
<td>OPR 7000</td>
</tr>
<tr>
<td>TAD 1000</td>
<td>NOP 7000</td>
</tr>
<tr>
<td>ISZ 2000</td>
<td>CLA 7200</td>
</tr>
<tr>
<td>DCA 3000</td>
<td>CLL 7100</td>
</tr>
<tr>
<td>JMS 4000</td>
<td>CMA 7040</td>
</tr>
<tr>
<td>JMP 5000</td>
<td>CML 7020</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input/Output Instructions</th>
<th>Group 2 Operate Instructions (microprogrammable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOT 6000</td>
<td>SMA 7500</td>
</tr>
<tr>
<td>ION 6001</td>
<td>SZA 7440</td>
</tr>
<tr>
<td>IOF 6002</td>
<td>SPA 7510</td>
</tr>
<tr>
<td>KSF 6031</td>
<td>SNA 7450</td>
</tr>
<tr>
<td>KCC 6032</td>
<td>SNL 7420</td>
</tr>
<tr>
<td>KRS 6034</td>
<td>SZL 7430</td>
</tr>
<tr>
<td>KR0 6036</td>
<td>SKP 7410</td>
</tr>
<tr>
<td>TSF 6041</td>
<td>OSR 7404</td>
</tr>
<tr>
<td>TCF 6042</td>
<td>HLT 7402</td>
</tr>
<tr>
<td>TPC 6044</td>
<td>LAS 7604</td>
</tr>
<tr>
<td>TLS 6046</td>
<td></td>
</tr>
</tbody>
</table>
22.2 Initially Defined Operations for DEC-338 PDPMAP

All of the initially defined operations for PDP-8 PDPMAP are included in those for DEC-338 PDPMAP. In addition, the following operations associated with the DEC-338 are defined.

**Input/Output Instructions**

- RPDP 6051
- RXP 6052
- RYP 6054
- RDAC 6061
- RS1 6062
- RS2 6064
- RPB 6071
- SPDP 6135
- SIC 6145
- LBF 6155
- INIT 6165
- RES1 6174
- RES2 6164
- CFD 6161
- STPD 6154
- SLP 6132
- SPES 6151
- SPEF 6152
- SPSF 6171
- SPMI 6172
### 22.3 DEC-330 Control State Commands (microprogrammable)

<table>
<thead>
<tr>
<th>Command</th>
<th>Operation</th>
<th>Microprogrammable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPOF</td>
<td>0040</td>
<td>PEDS 3001</td>
</tr>
<tr>
<td>LPON</td>
<td>0060</td>
<td>PNI 3002</td>
</tr>
<tr>
<td>SC1</td>
<td>0400</td>
<td>PNLS 3004</td>
</tr>
<tr>
<td>SC2</td>
<td>0500</td>
<td>PNM 3010</td>
</tr>
<tr>
<td>SC4</td>
<td>0600</td>
<td>* SK1 4000</td>
</tr>
<tr>
<td>SC8</td>
<td>0700</td>
<td>** INV 0400</td>
</tr>
<tr>
<td>INT</td>
<td>0010</td>
<td>** CLAT 0200</td>
</tr>
<tr>
<td>EDS</td>
<td>1001</td>
<td>** COAT 0100</td>
</tr>
<tr>
<td>CCB</td>
<td>1002</td>
<td>* SK2 5000</td>
</tr>
<tr>
<td>CSB</td>
<td>1004</td>
<td>* SK3 6000</td>
</tr>
<tr>
<td>POINT</td>
<td>1100</td>
<td>* SK4 6100</td>
</tr>
<tr>
<td>INCR</td>
<td>1110</td>
<td>SKIP 6240</td>
</tr>
<tr>
<td>VEC</td>
<td>1120</td>
<td>SNSZ 6220</td>
</tr>
<tr>
<td>VECON</td>
<td>1130</td>
<td>SPBI 6210</td>
</tr>
<tr>
<td>SVEC</td>
<td>1140</td>
<td>SPB2 6204</td>
</tr>
<tr>
<td>CHAR</td>
<td>1150</td>
<td>SLPSI 6202</td>
</tr>
<tr>
<td>GRAPH</td>
<td>1160</td>
<td>CLPSI 6201</td>
</tr>
<tr>
<td>CLDF</td>
<td>1200</td>
<td>SCUP 6340</td>
</tr>
<tr>
<td>STOP</td>
<td>1400</td>
<td>SCDN 6360</td>
</tr>
<tr>
<td>*** JUMP</td>
<td>2000</td>
<td>INTUP 6310</td>
</tr>
<tr>
<td>*** PJMP</td>
<td>2010</td>
<td>INTDN 6314</td>
</tr>
<tr>
<td>* POP</td>
<td>3000</td>
<td>BKON 6302</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BKOF 6301</td>
</tr>
</tbody>
</table>

* usable only in operation field

** usable only in variable field

*** specially defined to create two words (see Sect. 7)
23. Sample Assembly

In order to illustrate the important features of the PDPMAP Assembly System, a DEC-338 program, called "RUBBAN", was written. An annotated assembly listing is provided followed by the PDPMAP Postprocessor output. Also, an explanation of highlights of the assembly and postprocessing is given. The program was specially designed to point out many assembly features, and it is therefore recommended that the reader carefully study the listing and output along with the explanation.

RUBBAN is a program which works on a standard DEC-338 with at least 8K memory locations. The program displays a square box on the screen which is used to track the light pen. A vector is drawn from the screen center to the center of the box, giving a rubber band effect as the box is moved around the screen by the user.

The reader who is not interested in DEC-338 programming may ignore the program content, but should still study the assembly features.

23.1 Listing

An assembly listing of the program RUBBAN is given here. The listing was produced by IBMAP on the 7040. The first column of the listing indicates PDP-8 locations (in octal) where code was assembled. The next 12 octal digits constitute the intermediate code produced by IBMAP. The next five binary digits are relocation information for the IBSYS loader and may be ignored. Next, the source symbolic card is listed with programmer's comments at the right. The last column of the listing indicates the statement number assigned by IBMAP.
COMMENT CARD

MACRO DEFINITION

OPERATION CODE

FULL PDP-8 ADDRESS

LINKING AREA IS 160-177

LITERALS ASSEMBLED HERE (160-163).

THEY ARE SHOWN AT THE END OF THE LISTING.
MICROPROGRAMMING WITH COMBINATION OF THREE

ABSORG 15C
LNSK

START CLA STL+RAR

CDF 8
DCA VECX
DATA FIELD = 1
INITIALIZE VECTOR TO 0,0

CC231 10CC 03 0 10605 10001
CCA VECY

CC232 00CC 00 0 06201 10000
CDF 0
DATA FIELD = 0

CC233 00CC 00 0 07301 10000
RSTART SETUNEN
LINK = 0, ACC. = 1

CC234 00CC 00 0 07006 10000
RTL
ACC. = 4

CC235 00CC 00 0 06145 10000
SIC
INHIBIT EDGE FLAGS

BINARY CARG CCC02

CC236 10CC 01 0 00162 10001
TAD
=04374

CC237 00CC 00 0 06155 10000
LBF

CC240 00CC 00 0 07200 10000
CLA

CC241 10CC 01 0 00163 10001
TAD

CC242 00CC 00 0 06165 10000
INIT

CC243 00CC 00 0 07200 10000
CLA

CC244 00CC 00 0 06171 10000
WAIT SPSF

CC245 10CC 05 0 00244 10001
JMP

CC246 00CC 00 0 06052 10000
RXP

CC247 10CC 03 0 00600 10001
CCA TEMP1

CC250 00CC 00 0 06054 10000
RYP

CC251 10CC 04 0 00401 10001
JMS COMSTU

CC252 00CC 00 0 10605 10001
PZE VECY

CC253 10CC 01 0 00603 10001
TAD TEMP1

CC254 10CC 24 0 0401 10001
JMS COMSTU

CC255 00CC 00 0 10606 10001
PZE VECX

CC256 00CC 00 0 06164 10000
RES2

CC257 10CC 25 0 00244 10001
JMP WAIT

OCTAL LITERAL

ACC. = 4400
BREAK FIELD = 1

CURRENT LOCATION

START DISPLAY AT DISFL

WAIT SPSF

G-1 WAIT FOR DISPLAY TO STOP

READ X COORDINATE
READ Y COORDINATE
SAVE X COORDINATE

COMPUTE Y COMPONENT AND STORE

GET X COORDINATE
COMPUTE X COMPONENT AND STORE

RESUME DISPLAY
LOOP TO KEEP TRACKING

LINKING OPTION TO FORCE
INITIAL PAGE LINKING
## IBMAP Assembly Language Rubban Page 4

### BINARY CAGE C0003

<table>
<thead>
<tr>
<th>PCINTR</th>
<th>RELORG</th>
<th>TEMPORARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
</tbody>
</table>

**FORM OF IOT & OPR INSTRUCTIONS**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC401</td>
<td>C0 C0 C0</td>
</tr>
<tr>
<td>CC402</td>
<td>10 C0 01</td>
</tr>
<tr>
<td>CC403</td>
<td>C0 C0 C0</td>
</tr>
<tr>
<td>CC404</td>
<td>10 C0 05</td>
</tr>
<tr>
<td>CC405</td>
<td>00 C0 00</td>
</tr>
<tr>
<td>CC406</td>
<td>10 C0 C0</td>
</tr>
<tr>
<td>CC407</td>
<td>10 C0 C3</td>
</tr>
<tr>
<td>CC410</td>
<td>10 C0 C0</td>
</tr>
<tr>
<td>CC411</td>
<td>10 C0 C0</td>
</tr>
</tbody>
</table>

### BINARY CAGE C0004

**INDIRECT ADDRESSING**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC412</td>
<td>00 C0 00</td>
</tr>
<tr>
<td>CC413</td>
<td>10 C0 01</td>
</tr>
<tr>
<td>CC414</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>CC415</td>
<td>10 C0 63</td>
</tr>
<tr>
<td>CC416</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>CC417</td>
<td>10 C0 02</td>
</tr>
<tr>
<td>CC420</td>
<td>10 C0 65</td>
</tr>
<tr>
<td>CC421</td>
<td></td>
</tr>
<tr>
<td>CC422</td>
<td></td>
</tr>
<tr>
<td>CC423</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIELD</th>
<th>PAGE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>126</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10600</td>
<td>C0 C0 C0</td>
</tr>
<tr>
<td>10601</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>10602</td>
<td>C0 C0 C0</td>
</tr>
<tr>
<td>10603</td>
<td>C0 C0 C0</td>
</tr>
<tr>
<td>10604</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>10605</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>10606</td>
<td>00 C0 C0</td>
</tr>
</tbody>
</table>

### BINARY CAGE C0005

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10607</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>10608</td>
<td>00 C0 C0</td>
</tr>
<tr>
<td>10611</td>
<td>00 C0 C0</td>
</tr>
</tbody>
</table>
The image contains a page from an IBM assembly language manual. The page features assembly code, macro calls, and expansion examples. The code uses hexadecimal notation for addressing and includes comments in the assembly language syntax. The page is part of a larger manual, indicated by the page number 5.
## REFERENCES TO DEFINED SYMBOLS

<table>
<thead>
<tr>
<th>VALUE</th>
<th>NAME</th>
<th>STATEMENT NUMBERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC163</td>
<td>CINIT</td>
<td>68</td>
</tr>
<tr>
<td>CC401</td>
<td>CMSTG</td>
<td>96, 99, 114, 121, 122</td>
</tr>
<tr>
<td>IC600</td>
<td>DISFIL</td>
<td>73, 144.0001, 144.0005</td>
</tr>
<tr>
<td>CC600</td>
<td>..CC1</td>
<td>51.0002</td>
</tr>
<tr>
<td>CC601</td>
<td>..CC2</td>
<td>52.0002</td>
</tr>
<tr>
<td>CC1CC</td>
<td>..CC3</td>
<td>64.0003</td>
</tr>
<tr>
<td>CCCCC</td>
<td>..CC4</td>
<td>64.0005</td>
</tr>
<tr>
<td>CC1CC</td>
<td>..CC5</td>
<td>64.0005</td>
</tr>
<tr>
<td>CC160</td>
<td>..CC6</td>
<td>70.0003</td>
</tr>
<tr>
<td>CC104</td>
<td>..CC7</td>
<td>70.0005</td>
</tr>
<tr>
<td>CC160</td>
<td>..CC10</td>
<td>70.0005</td>
</tr>
<tr>
<td>CC16C</td>
<td>..CC11</td>
<td>71.0003</td>
</tr>
<tr>
<td>CC164</td>
<td>..CC12</td>
<td>75.0004</td>
</tr>
<tr>
<td>CC226</td>
<td>..CC13</td>
<td>75.0004</td>
</tr>
<tr>
<td>CC226</td>
<td>..CC14</td>
<td>76.0003</td>
</tr>
<tr>
<td>CC457</td>
<td>..CC16</td>
<td>104.0006</td>
</tr>
<tr>
<td>CC260</td>
<td>..CC17</td>
<td>104.0008</td>
</tr>
<tr>
<td>CC400</td>
<td>..CC20</td>
<td>104.0008</td>
</tr>
<tr>
<td>CC400</td>
<td>..CC21</td>
<td>105.0005</td>
</tr>
<tr>
<td>CC401</td>
<td>..CC22</td>
<td>105.0005</td>
</tr>
<tr>
<td>CC620</td>
<td>..CC23</td>
<td></td>
</tr>
<tr>
<td>CC421</td>
<td>..CC24</td>
<td>127.0010</td>
</tr>
<tr>
<td>IC600</td>
<td>..CC25</td>
<td>127.0010</td>
</tr>
<tr>
<td>IC600</td>
<td>..CC27</td>
<td>144.0004</td>
</tr>
<tr>
<td>IC642</td>
<td>..CC31</td>
<td>145.0003</td>
</tr>
<tr>
<td>CC000</td>
<td>LCC.C</td>
<td>51.0002, 52.0002, 64.0003, 70.0003, 75.0002, 104.0001, 104.0006, 105.0006</td>
</tr>
<tr>
<td>OC4CC</td>
<td>PCINTR</td>
<td>115, 119</td>
</tr>
<tr>
<td>OC233</td>
<td>RSTART</td>
<td>68</td>
</tr>
<tr>
<td>OC226</td>
<td>START</td>
<td>67</td>
</tr>
<tr>
<td>IC600</td>
<td>TEMP1</td>
<td>94.98</td>
</tr>
<tr>
<td>OC601</td>
<td>TEMP2</td>
<td>113.117</td>
</tr>
<tr>
<td>IC606</td>
<td>VECX</td>
<td>75.100</td>
</tr>
<tr>
<td>IC605</td>
<td>VECY</td>
<td>80.97</td>
</tr>
<tr>
<td>OC244</td>
<td>WAIT</td>
<td>1C2</td>
</tr>
<tr>
<td>OC160</td>
<td>P.CCCC</td>
<td>72, 145.0006</td>
</tr>
<tr>
<td>CC163</td>
<td>P.CCC1</td>
<td>72</td>
</tr>
<tr>
<td>CC162</td>
<td>P.CCC2</td>
<td>85</td>
</tr>
<tr>
<td>CC160</td>
<td>P.CCC3</td>
<td>108</td>
</tr>
<tr>
<td>OC161</td>
<td>P.CCC4</td>
<td>112</td>
</tr>
<tr>
<td>4C011</td>
<td>P.CCC5</td>
<td>145.0012</td>
</tr>
</tbody>
</table>

## REFERENCES TO LOCATION COUNTERS

<table>
<thead>
<tr>
<th>LC START</th>
<th>NAME</th>
<th>STARTING AND ENDING STATEMENT NUMBERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCCCC</td>
<td></td>
<td>1-64.0003, 64.0006-70.0003, 70.0006-71.0001, 71.0004-75.0002, 75.0004-104.0009-105.0003, 105.0006-127.0008, 127.0011-145.001</td>
</tr>
<tr>
<td>4C000</td>
<td></td>
<td>64.0004-64.0006, 70.0004-70.0005, 70.0006-71.0003, 75.0003-75.0004, 104.0007-104.0008, 105.0004-105.0005, 127.0009-127.0010, 145.0002-145.0003</td>
</tr>
<tr>
<td>PAGE</td>
<td>LINKED INSTRUCTION ADDRESS</td>
<td>LINK WORD ADDRESS</td>
</tr>
<tr>
<td>------</td>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>00102</td>
<td>06164</td>
</tr>
<tr>
<td></td>
<td>00103</td>
<td>00165</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>00230</td>
<td>00255</td>
</tr>
<tr>
<td></td>
<td>00231</td>
<td>00252</td>
</tr>
<tr>
<td></td>
<td>00247</td>
<td>00163</td>
</tr>
<tr>
<td></td>
<td>00251</td>
<td>00260</td>
</tr>
<tr>
<td></td>
<td>00253</td>
<td>00163</td>
</tr>
<tr>
<td></td>
<td>00254</td>
<td>00166</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00200</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>00407</td>
<td>00167</td>
</tr>
<tr>
<td></td>
<td>00413</td>
<td>00167</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FINAL STATUS OF THE INITIAL PAGE OF FIELD 0**

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>PAGE LINKAGE</th>
<th>CURRENT PAGE ADDRESS</th>
<th>INITIAL PAGE ADDRESS</th>
<th>LINK VALUE</th>
<th>UNUSED LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
23.2 Explanation

ASSEMBLY

In the above assembly listing of RUBBAN the reader should note:

1. The 49 cards composing the DEC-338 PDPMAP DEFINITIONS were included at the beginning of the source deck, but only the first card is listed.

2. The program includes the definition and use of a new Operate instruction mnemonic. The OCTADR pseudo-operation was used twice.

3. A macroinstruction was defined and was used four times.

4. The four origin pseudo-operations were used: OCTORG, ABSORG, RELORG, and PAGE (with and without an argument). Also the FIELD pseudo-operation was used.

5. Positive and negative octal literals were used and pooled together on page zero by the LORG pseudo-operation.

6. Operate instructions and DEC-338 commands were used alone and in double and triple combinations.

7. Input/Output instructions were used alone, and, in the case of CDF, a decimal number was included to form the proper instructions.

8. The uses of "*" as comment card indicator, indirect addressing indicator, and current location indicator were all included.

9. Examples of the PZE and OCT data generation pseudo-operations are given.

10. There are some off-page memory references, uses of the LINKS pseudo-operation, and an example of a linking option.
CROSS-REFERENCE DICTIONARY

The page following the assembly listing is the Cross-Reference Dictionary of RUBBAN produced by IBMAP. It is a symbol table in lexicographic order along with the numbers of the statements where each symbol was referenced.

When a symbol table is obtained, all symbols listed in the Cross-Reference Dictionary will be punched by SYMTAB except:

1. LOC.0
2. Symbols of the form ..nnnn (created symbols)
3. Symbols of the form P.nnnn (literal symbols)

POSTPROCESSING (LINKING)

As a result of the assembly of RUBBAN, two linking areas were specified through uses of the LINKS pseudo-operation:

AREA A: locations 160 - 177. Note that locations 160 - 163 can be used for linking, and locations 164 - 177 can be used for the creation of links.

AREA B: locations 226 - 377. Note that those locations between 226 and 257 where memory reference instructions were not assembled can be used for linking, and locations 260 - 377 can be used for the creation of links.

A step-by-step account of the linking procedure is given below: The reader is advised to follow the PDPMAP Postprocessor output in parallel with the explanation.

1) The first instruction which requires a link is at location 102, where IBMAP assembled a JMP instruction addressing location 226. Since no location in the linking area A already contains 0226, a link must be created, and the first possible link word is 164.
Therefore the instruction in location 102 is made into an indirect JMP instruction addressing location 164, which in turn contains 0226.

2) Similarly, a link is created for the JMP instruction in location 103.

3) The summary for the page indicates that two links were created for instructions on the first page and linking area A has 10 (decimal) words available for the creation of further links.

4) The next instruction requiring a link is at location 230, where there is a cross-field DCA instruction addressing location 10606. Since the linking area B includes a location containing 0606, no link creation is necessary. Instead, location 255 is used as a link by forming at location 230 an indirect DCA instruction addressing location 255. The PDFMAP Postprocessor listing includes the number 0606 in parentheses to indicate an already existing link word was used.

5) Similarly, a link was used at location 252 for the DCA instruction in location 231.

6) Location 247 contains the next memory reference instruction which requires linking. Since there is no existing link word on the current page, an attempt is made to use a link on the initial page. The Postprocessor finds that there is such a link work in location 163, and therefore it is not necessary to create one.

7) The JMS instruction at location 251 causes a link to be created on the current page. The link is placed at the first available location in linking area B.

8) As at location 247, an existing initial page link is used at location 253.

9) The instruction assembled at location 254 is an example of a user-controlled linking option. If the option had not been included, the Postprocessor would have used the link previously created on the current page. The option, however, forces initial page linking and a link is created on the initial page.
10) The summary of results of the current page are listed. Note that the number of links created represents only those links created on the current page (in linking area B).

11) There is no linking area on the same page as location 407, and therefore a link is created on the initial page.

12) The link created above is used by the instruction at location 413.

13) There are no summary statistics listed for this page since no linking area was specified.

14) A summary for the field is then listed indicating the final number of unused locations in the linking area on the initial page. This number is two less than the number originally listed for that linking area since two links were created as the Postprocessor proceeded through the field.

15) Since there are no further occurrences of the LINKS pseudo-operation, no more summaries are included. A page starting address is printed for each page where code was assembled. In addition, the initial address of a memory field is always listed if there is assembled code somewhere on that field.
REFERENCES


This report describes the PDPMAP Assembly System which is used to assemble symbolic programs written for a Digital Equipment Corporation PDP-8 or DEC-338 with up to 16-K memory locations. The system is implemented at the University of Pennsylvania on an IBM 7040 and DEC PDP-8 connected by a high-speed data channel (IBM 7904 and DEC DMS). The PDPMAP System uses the powerful assembler of a larger computer (IBM 7040 MAP Assembler) to quickly assemble programs for a small computer.
<table>
<thead>
<tr>
<th>KEY WORDS</th>
<th>LINK A</th>
<th>LINK B</th>
<th>LINK C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly System</td>
<td>ROLE</td>
<td>WT</td>
<td>ROLE</td>
</tr>
<tr>
<td>Microprogramming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microinstructions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off-page linking</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Security Classification