

# DAP MANUAL

## FOR 16-BIT DDP COMPUTERS

|       |                 |      |          |
|-------|-----------------|------|----------|
| 00255 | 101400          | SMI  |          |
| 00256 | 0 01 00265      | JMP  | POSA     |
| 00257 | -0 10 00000     | CALL | TWOS     |
| 00260 | 0 04 00325      | STA  | XH       |
| 00261 | 000201          | IAB  |          |
| 00262 | 0 04 00326      | STA  | XL       |
| 00263 | 0 02 00347      | LDA  | - - 1    |
| 00264 | 0 01 00271      | JMP  | POSB     |
| 00265 | 0 04 00325 POSA | STA  | XH       |
| 00266 | 000201          | IAB  |          |
| 00267 | 0 04 00326      | STA  | XL       |
| 00270 | 140040          | CRA  |          |
| 00271 | 0 04 00336 POSB | STA  | COMM + 2 |
| 00272 | 0 02 00254      | LDA  | POS      |
| 00273 | 0 07 00350      | SUB  | - 2      |
| 00274 | 0 04 00342      | STA  | RETN     |
| 00275 | -0 02 00342     | LDA* | RETN     |
| 00276 | 0 04 00342      | STA  | RETN     |
| 00277 | -0 02 00342     | LDA* | RETN     |
| 00300 | 0 04 00334      | STA  | COMM     |
| 00301 | 0 06 00345      | ADD  | - 1      |
| 00302 | 0 04 00335      | STA  | COMM + 1 |
| 00303 | -0 02 00335     | LDA* | COMM - 1 |
| 00304 | 000201          | IAB  |          |
| 00305 | -0 02 00334     | LDA* | COMM     |
| 00306 | 101400          | SMI  |          |
| 00307 | 0 01 00316      | JMP  | POSC     |
| 00310 | -0 10 00000     | CALL | TWOS     |
| 00311 | 0 04 00327      | STA  | YH       |
| 00312 | 000201          | IAB  |          |
| 00313 | 0 04 00330      | STA  | YL       |
| 00314 | 140040          | CRA  |          |
| 00315 | 0 01 00322      | JMP  | POSD     |
| 00316 | 0 04 00327 POSC | STA  | YH       |
| 00317 | 000201          | IAB  |          |
| 00320 | 0 04 00330      | STA  | YL       |
| 00321 | 0 02 00347      | LDA  | - - 1    |
| 00322 | 0 06 00336 POSD | ADD  | COMM + 2 |
| 00323 | 0 04 00336      | STA  | COMM + 2 |

# Honeywell

COMPUTER CONTROL DIVISION

*Evolution*

**DAP-16 MANUAL**  
for the  
**DDP-116, DDP-416, and DDP-516**  
**General Purpose Computers**

December 1966

**Honeywell**

COMPUTER CONTROL DIVISION

*van der ...*

**COPYRIGHT 1966 by Honeywell Inc., Computer Control Division,  
Framingham, Massachusetts. Contents of this publication may not be  
reproduced in any form in whole or in part, without permission of the  
copyright owner. All rights reserved.**

**Printed in U.K. COPIA PRODUCTUM, HARROW.**

## CONTENTS

### Page

### SECTION I INTRODUCTION

|                    |     |
|--------------------|-----|
| Purpose of DAP-16  | 1-1 |
| Modes of Operation | 1-2 |
| Assembly Process   | 1-6 |
| Two-Pass Assembly  | 1-6 |
| One-Pass Assembly  | 1-7 |

### SECTION II THE DAP-16 LANGUAGE

|                        |     |
|------------------------|-----|
| Source Language Format | 2-1 |
| Location Field         | 2-2 |
| Operation Field        | 2-2 |
| Variable Field         | 2-2 |
| Comments Field         | 2-2 |
| DAP-16 Symbology       | 2-3 |
| Symbols                | 2-3 |
| Expressions            | 2-4 |
| Literals               | 2-4 |
| Asterisk Conventions   | 2-5 |
| Assembly Listing       | 2-5 |

### SECTION III DAP-16 PSEUDO-OPERATION

|  |      |
|--|------|
| Assembly Controlling Pseudo-Operations | 3-1  |
| ABS Pseudo-Operation                   | 3-2  |
| CFx Pseudo-Operation                   | 3-2  |
| END Pseudo-Operation                   | 3-2  |
| FIN Pseudo-Operation                   | 3-3  |
| LOAD Pseudo-Operation                  | 3-3  |
| MOR Pseudo-Operation                   | 3-4  |
| ORG Pseudo-Operation                   | 3-4  |
| REL Pseudo-Operation                   | 3-5  |
| Data Defining Pseudo-Operations        | 3-5  |
| BCI Pseudo-Operation                   | 3-5  |
| DAC Pseudo-Operation                   | 3-6  |
| DEC Pseudo-Operation                   | 3-6  |
| DBP Pseudo-Operation                   | 3-8  |
| OCT Pseudo-Operation                   | 3-10 |

## CONTENTS (Cont)

|                                      | <u>Page</u> |
|--------------------------------------|-------------|
| Loader-Controlling Pseudo-Operations | 3-10        |
| EXD Pseudo-Operation                 | 3-11        |
| LXD Pseudo-Operation                 | 3-11        |
| SETB Pseudo-Operation                | 3-11        |
| List-Controlling Pseudo-Operations   | 3-12        |
| EJCT Pseudo-Operation                | 3-12        |
| LIST Pseudo-Operation                | 3-12        |
| NLST Pseudo-Operation                | 3-12        |
| CALL Pseudo-Operation                | 3-13        |
| XAC Pseudo-Operation                 | 3-14        |
| SUBR Pseudo-Operation                | 3-14        |
| Storage Allocation Pseudo-Operations | 3-17        |
| BES Pseudo-Operation                 | 3-17        |
| BSS Pseudo-Operation                 | 3-18        |
| BSZ Pseudo-Operation                 | 3-19        |
| COMN Pseudo-Operation                | 3-19        |
| Symbol Defining Pseudo-Operation     | 3-20        |
| EQU Pseudo-Operation                 | 3-20        |
| Special Mnemonic Codes               | 3-20        |

## SECTION IV DAP-16 OPERATING INSTRUCTIONS

|                            |     |
|----------------------------|-----|
| Source Program Preparation | 4-1 |
| Paper Tape                 | 4-1 |
| Cards                      | 4-1 |
| Object Program Preparation | 4-2 |
| Error Diagnosis            | 4-3 |
| Object Program Format      | 4-4 |

## SECTION V PROGRAMMING EXAMPLES

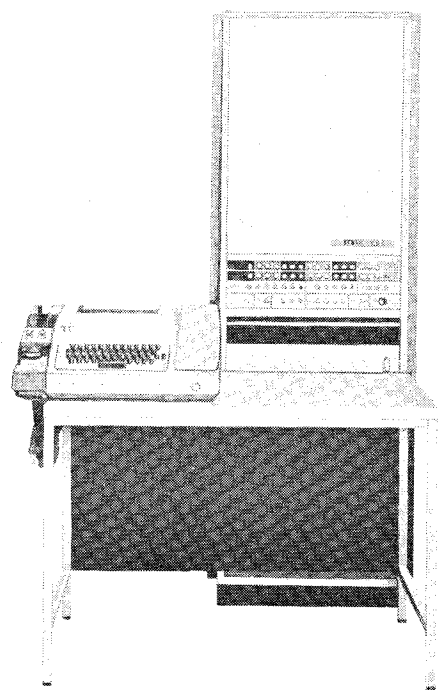
|   |     |
|---|-----|
| Appendix A Summary of DAP Pseudo-Operations           | 5-1 |
| Appendix B DAP Operation Codes                        | A-1 |
| Appendix C DDP-116 and DDP-516 Option Operation Codes | B-1 |
|   | C-1 |

## ILLUSTRATIONS

|                                  | <u>Page</u> |
|----------------------------------|-------------|
| 1-1 Desectorized Program Loading | 1-3         |
| 1-2 Processing of One Line       | 1-7         |
| 2-1 Assembly Listing             | 2-3         |
| 3-1 Floating-Point Formats       | 3-9         |

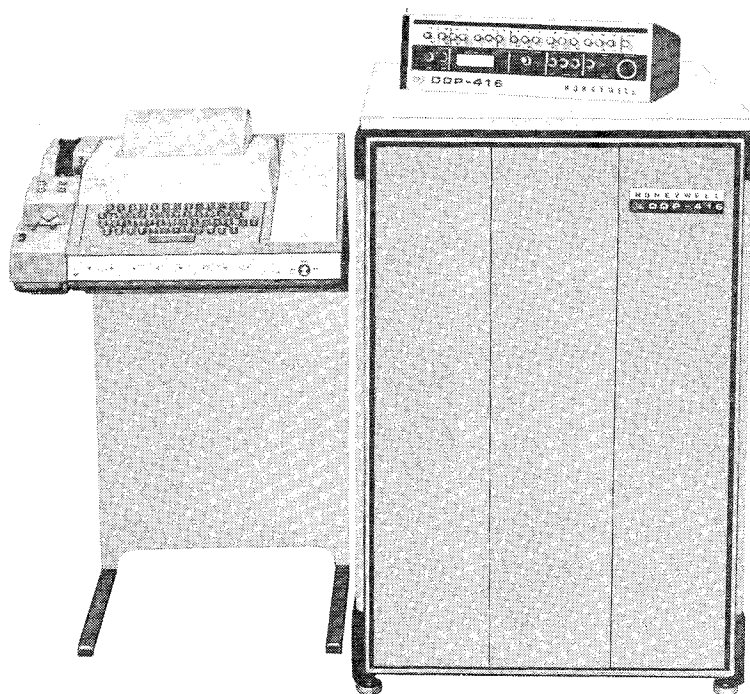
## TABLES

|  | <u>Page</u> |
|--|-------------|
| 3-1 Floating-Point Number Translations               | 3-9         |
| 4-1 A-Register Bit Settings For I/O Device Selection | 4-2         |



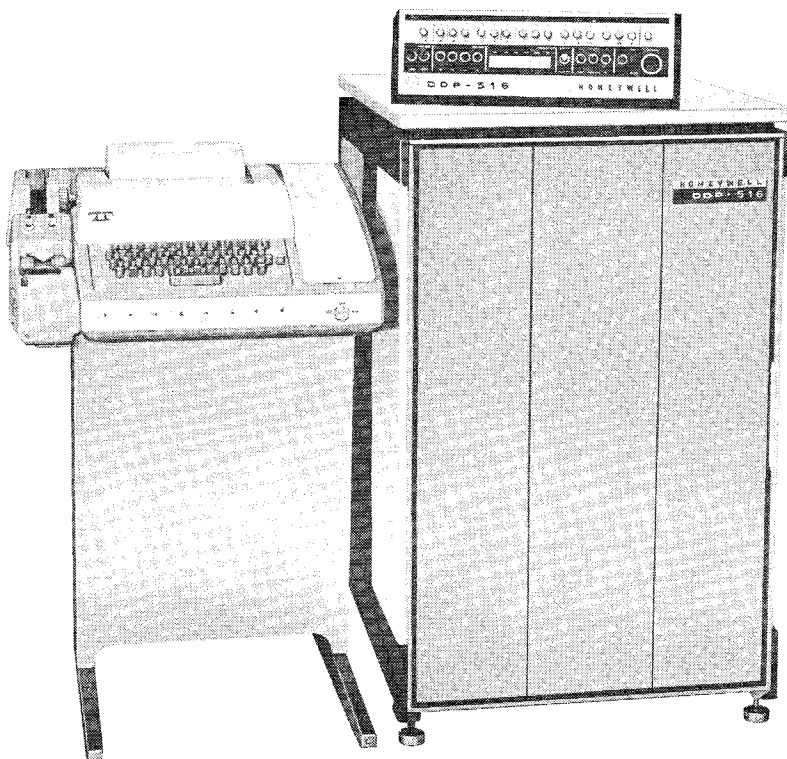
714

DDP-116  
GENERAL PURPOSE COMPUTER



3766

DDP-416  
GENERAL PURPOSE COMPUTER



3775

DDP-516  
GENERAL PURPOSE COMPUTER

## SECTION I INTRODUCTION

This manual describes the programming of the DDP-116, DDP-416 and DDP-516 General-Purpose Digital Computers using the DAP-16 symbolic assembly program. Included are discussions of the DAP-16 language, DAP-16 pseudo operations, and DAP-16 programming techniques. For a complete discussion of symbolic language programming and computer characteristics, refer to the DDP-116, DDP-416, or the DDP-516 Programmers Reference Manuals.

### PURPOSE OF DAP-16

DAP-16 is a programming aid that translates a symbolic (source) program into machine language (object) code. DAP-16 provides the following features.

- a. Enables symbolic programming while maintaining the characteristics, flexibility, speed, and conciseness of machine language programming.
- b. Permits the assignment of symbolic addresses to storage locations.
- c. Provides numerous pseudo-operations to supplement the standard DDP-116, DDP-416, and DDP-516 instruction repertoires.

The pseudo-operations allow the programmer to express concepts that have no counterpart in machine language. Among the capabilities of the pseudo-operations are programmer defined assembly and loader controls, data definitions and program linkages.

DAP-16 incorporates the following features.

- a. Employs an input/output selector (IOS) concept for input/output device selection. (Preselected input/output for machines with less than 8K of memory.)
- b. Provides a pool table for storage of symbols and literals, thereby avoiding fixed-length tables.
- c. Allows alphanumeric literals.
- d. Allows compound expressions in the variable field.
- e. Prints out and assigns storage for undefined symbols.
- f. Flags illegal instructions and coding errors.
- g. Allows single- or double-precision fixed- or floating-point constants.
- h. Assembles DDP-116, DDP-416, or DDP-516 programs on any of the computers.
- i. Allows operation in either a one-pass or two-pass mode.
- j. Assembles programs which take advantage of the extended addressing, memory lockout, memory parity and double-precision arithmetic options.

## MODES OF OPERATION

DAP-16 operates in two basic modes: LOAD and DESECTORIZING. When operating in the load mode, DAP-16 closely approximates the addressing structure of a DDP-116, DDP-416, or a DDP-516 computer. Operand addresses must be within the same sector as the instruction, or in sector zero, otherwise an error flag is generated. This means that the programmer must be aware of an operand's location with respect to sector boundaries. Programs assembled in the LOAD mode are always absolute. It is the programmers responsibility to provide all indirect linkage required for intersector addressing and subroutine calls.

In the DESECTORIZING mode (Figure 1-1), DAP treats the DDP-16 class computer as if all of memory (up to 32K with the DDP-516 extended addressing option) is directly addressable. The DESECTORIZING mode does not require the programmer to be concerned with the location of the operands with respect to sector boundaries. It also makes possible the writing of very efficient, completely relocatable programs. In the DESECTORIZING mode, an extended object code is generated which provides the DAP/FORTRAN relocating loader with sufficient information to determine whether indirect address linkage must be supplied for any memory referencing instruction or whether the address may be inserted directly into the memory address instruction. Section IV contains a description of the extended object format used by the DAP/FORTRAN relocating loader to load desectorized programs. Large programs, when assembled in the DESECTORIZING mode, will generally require less memory space and operate faster because the tedious chore of defining and minimizing indirect address links is done by DAP-16 and the loader rather than by the programmer. In the DESECTORIZING mode, subroutines can be called using the CALL pseudo-operator, and all subroutine linkage will automatically be completed by the DAP/FORTRAN relocating loader. Programs may be assembled in the DESECTORIZING mode by placing an ABS pseudo-op (in the case of absolute programs) or a REL pseudo-op (in the case of relocatable programs) at the beginning of the program to be assembled. Programs written to be assembled in the DESECTORIZING mode should not, in general, modify or move memory referencing instructions within the program during the course of program execution. The common practice of address modification may be easily and safely accomplished by making the address to be modified an indirect address link (using the DAC pseudo-operation). This indirect address link may then be modified in the desired manner.

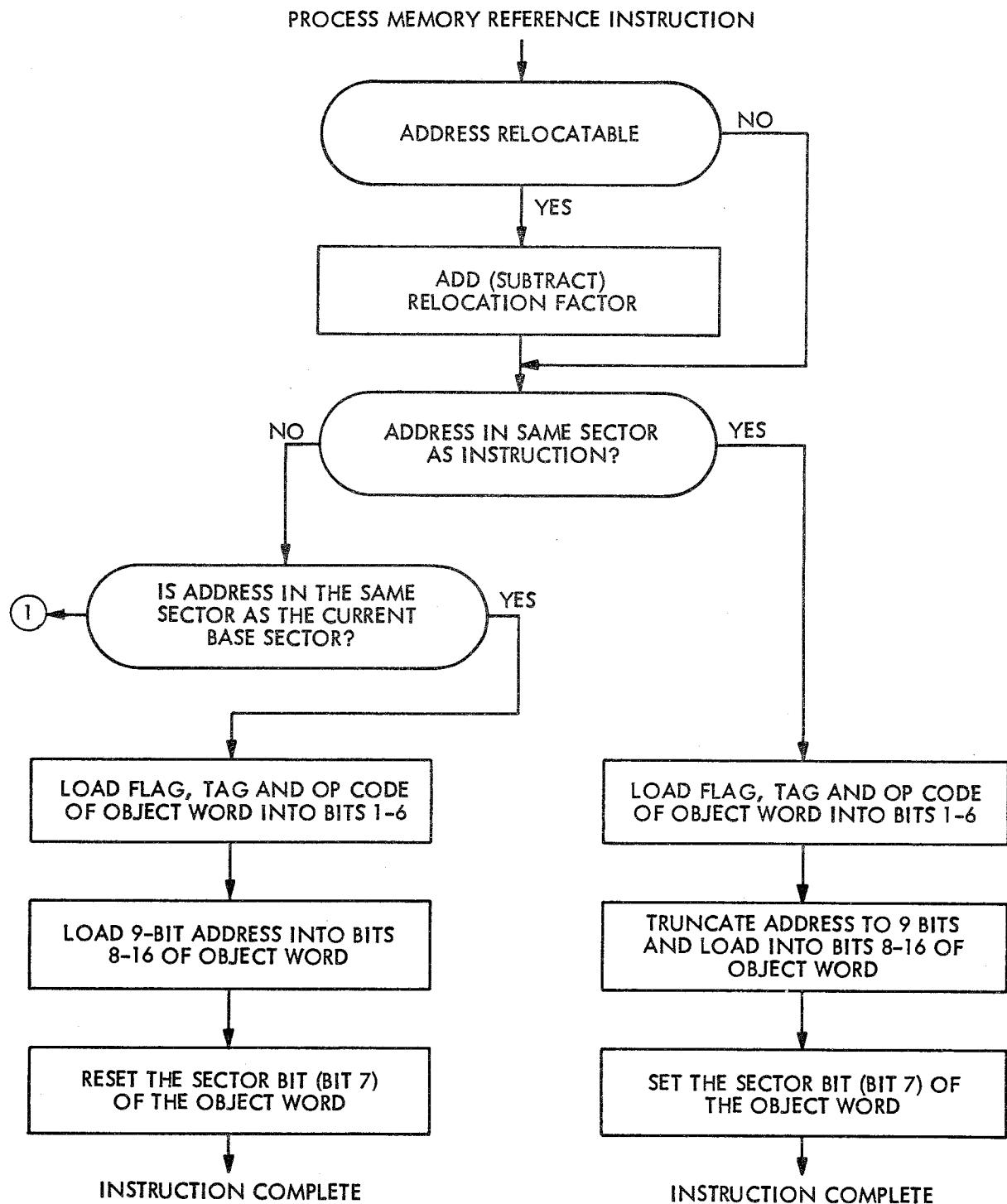


Figure 1-1. DESECTORIZED Program Loading (Part 1)

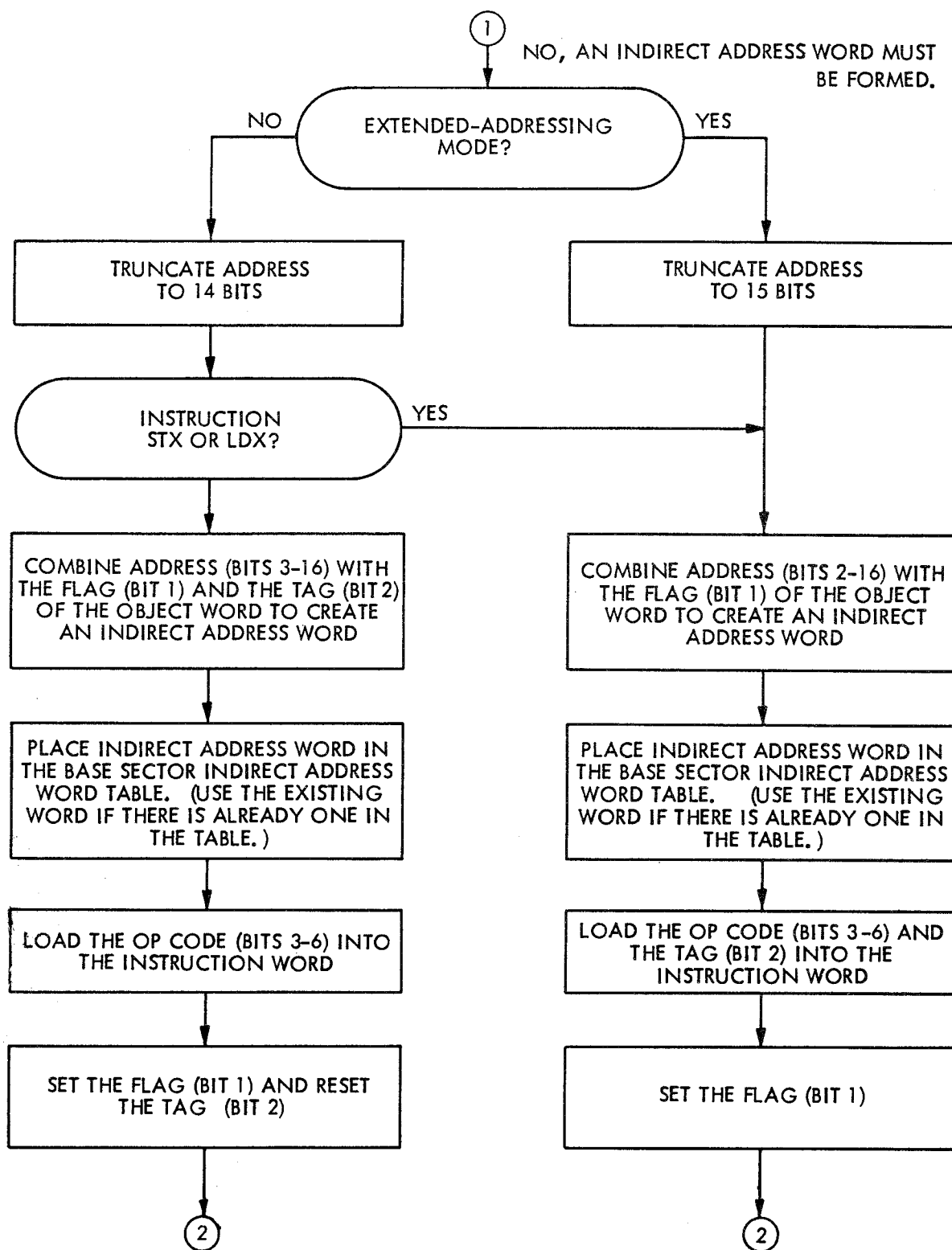


Figure 1-1. DESECTORIZED Program Loading (Part 2)

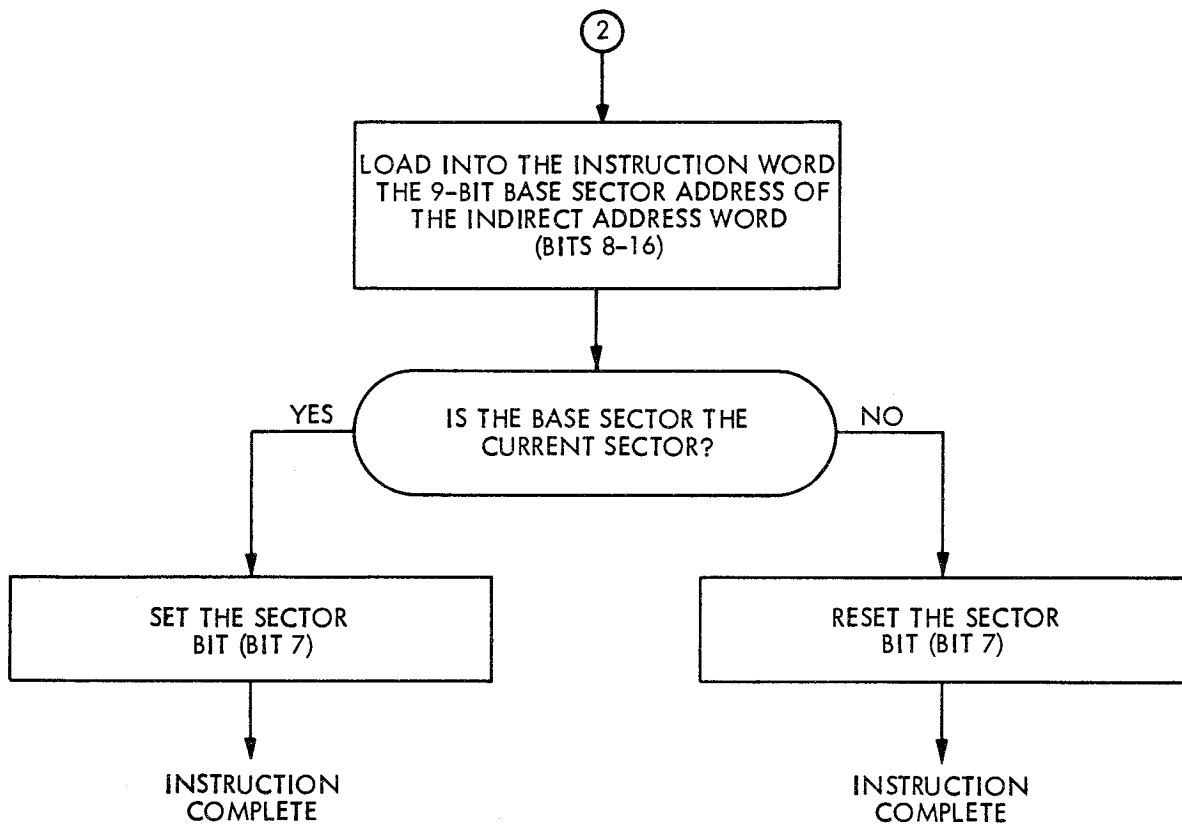


Figure 1-1. DESECTORIZED Program Loading (Part 3)

### ASSEMBLY PROCESS

Initially, the DAP-16 assembler is loaded into computer memory. The sequence of symbolic instructions in the source program to be are examined once or twice by DAP-16 at the programmers option. The contents of the A-register controls the number of passes and also the input/output device selection. If bit 1 (the sign bit) of the A-register is set to a 1, the two-pass mode is selected; if bit 1 is set to a 0, the one-pass mode is selected. The significance of the remaining 15 bits is discussed in Section IV.

### Two-Pass Assembly

The sequence of symbolic instructions in the source program to be assembled are examined twice by DAP-16; once to develop a dictionary of symbols, and a second time to assemble the object program by referencing the dictionary. The DAP-16 dictionary has storage space for defining operation mnemonics and symbols. Three cells are used for each operation or symbol; the encoded symbol or operation mnemonic is stored in the first two cells, and defined in the third. For machine instructions, the definition cell contains the

corresponding operation code. For location names, the definition cell contains the address at which the symbol is defined. For pseudo-operations, the definition cell contains a DAC to the location of the pseudo-operation analyzer in DAP-16. DAP-16 obtains locations for symbols by stepping a location counter for each line of the source program.

Program assembly takes place during pass two. Printing of each line is completed before the next line is started, reducing requirements for storage space. The line is read from the tape or card, stored in a special buffer (part of memory), the instruction or data word assembly is performed and, if requested, the assembled line is printed. Punching of the object program and punching or printing of the assembly listing is under control of the contents of the A-register.

Figure 1-2 illustrates how each line is processed. DAP-16 calls the subroutines necessary for reading and storing one line of type. The line is separated into its constituent fields, and the operation mnemonic is examined. The nature of the indicated operation (normal or pseudo) determines the subroutines to be called to process the operation field. For normal operations, DAP-16 determines the specified machine operation by table look-up and then places the operation-code in the appropriate portion of the instruction word being assembled. For pseudo-operations, analytical subroutines are called, and serve to modify the assembly process, allocate storage, define data words, or provide for program linkages at load time.

The variable field is then processed. Alphanumeric, octal, and decimal information is converted to binary; the DAP-16 dictionary is then searched to evaluate symbols, and calculations are performed to evaluate expressions. If the operation field specified a normal machine operation, the resultant value forms the address field of the instruction being assembled.

### One-Pass Assembly

The dictionary development and the object program assembly is accomplished in the same pass in a one-pass assembly. Forward referenced symbols (those that are used before being defined) have an unknown value at the assembly time. DAP-16 flags such symbols with a double asterisk (\*\*) and assigns each symbol an internal symbol number which is outputted with the instruction in which the symbol occurs. The loader program maintains a table of symbol numbers and their use. When the value of the symbol becomes known, DAP-16 outputs the value along with the object program so that the loader can fill in references to the symbol. The object program resulting from a one-pass assembly is longer than that for a two-pass assembly because of the additional information that must be supplied to the loader. Programs assembled in the one-pass mode must be loaded by the extended version of the DAP/FORTRAN loader (LDR) rather than the standard loader (SLDR).

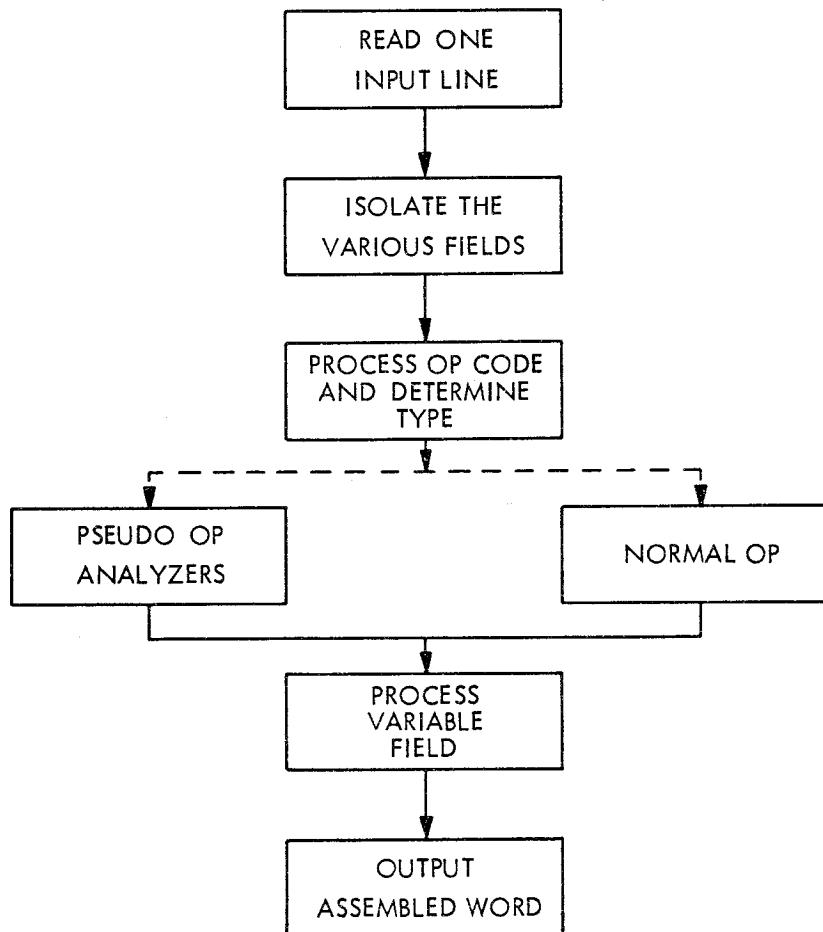


Figure 1-2. Processing of One Line



## SECTION II THE DAP-16 LANGUAGE

This section describes the format and symbology of the source language to be used with the DAP-16 assembly program. A discussion of the DAP-16 assembly listing is included in this section to show the correlation of source and object programs in preparation for the discussion of DAP-16 pseudo-operations in Section III.

### SOURCE LANGUAGE FORMAT

Programs written in the DAP-16 source language consist of a sequence of symbolic instructions or statements known as source lines. The example below shows a typical symbolic instruction written on a DAP-16 coding form. This instruction represents one source line.

|            |   |           |      |            |               |
|------------|---|-----------|------|------------|---------------|
| PROGRAMMER |   |           | DATE |            | PAGE          |
| PROGRAM    |   |           |      |            | CHARGE        |
| LOCATION   | ① | OPERATION | ①    | ADDRESS, X | ① COMMENTS    |
| 1          | 4 | 6         | 10   | 12         | 30 72         |
| STRT       |   | LDA       |      | CØNS       | LØAD CØNSTANT |

As indicated in the coding sheet, symbolic instructions consist of four fields as follows.

- a. The LOCATION field occupying character positions 1 through 4 of the source line.
- b. The OPERATION field occupying character positions 6 through 10 of the source line.
- c. The VARIABLE field beginning at character position 12 and continuing until a blank character or column 72 is present. This field is subdivided into the address subfield and index subfield. The address and index subfields are separated by a comma.
- d. The COMMENTS field begins at the character following the first blank character which terminates the VARIABLE field.

The example above shows an instruction which is located at the symbolic location STRT. The effect of the instruction is to load a constant, located at the symbolic location CONS, into the A-register. The comments field has no effect on the program. The significance of the several fields are discussed in more detail in the paragraphs that follow.

### Location Field

The location field may be used to assign a symbolic address or "label" to an instruction so that the instruction can be referred to elsewhere in the program. The symbolic address in the location field consists of one to four characters, at least one of which is non-numeric. DAP-16 assigns memory addresses to the symbolic locations when assembling the object program.

### Operation Field

The operation field is analogous to the operation-code portion of a machine language instruction. The contents of the operation field may be either a machine language instruction mnemonic, or one of the pseudo-operation mnemonics provided in the DAP-16 repertoire. Operation mnemonics are either three or four characters in length. In addition to specifying an operation, the operation field may also specify that indirect addressing is desired by writing an asterisk (\*) immediately following the operation-code mnemonic. The DDP-116, DDP-416, and DDP-516 Programmers Reference Manuals specify the machine language instruction mnemonic and the function of each DDP-16 class instruction.

### Variable Field

The variable field is normally used to specify an address and index register for DDP-16 class instructions. When used with a DAP-16 pseudo-operation, the significance of the variable field depends upon the nature of the pseudo-operation. (Pseudo-operations are discussed in Section III.)

### Comments Field

The comments field may be used for any comments the programmer cares to write. This field has no effect on the assembler, but it is printed out on the symbolic assembly listing. The format of the assembly listing is shown in Figure 2-1.

The portion of the assembly listing appearing on the right is a copy of the original source program input.

|   |      |       |             | *SAMPLE ASSEMBLY LISTING |         |
|---|------|-------|-------------|--------------------------|---------|
|   | 0001 |       |             |                          |         |
|   | 0002 |       |             | ORG                      | 512     |
|   | 0003 | 01000 | 0 02 01001  | STRT LDA                 | *+1     |
|   | 0004 | 01001 | 0 04 01000  | STA                      | *-1     |
| A | 0005 | 01002 | -0 02 00000 | LDA*                     |         |
|   | 0006 | 01003 | 0 06 01010  | ADD                      | =15     |
|   | 0007 | 01004 | 0 06 01011  | ADD                      | =15     |
|   | 0008 | 01005 | 0 04 00700  | STA                      | STRT-64 |
|   | 0009 | 01006 | 0 02 01012  | LDA                      | =-5     |
|   | 0010 | 01007 | 0414 76     | LGL                      | 2       |
|   |      | 01010 | 000017      |                          |         |
|   |      | 01011 | 000015      |                          |         |
|   |      | 01012 | 177773      |                          |         |
|   | 0011 |       |             | END                      | **      |

Figure 2-1. Assembly Listing

## DAP-16 SYMBOLOGY

In addition to operation and pseudo-operation mnemonics, the DAP-16 language contains symbols, expressions, and literals. A number of rules, discussed below, govern the formation and usage of these language elements.

### Symbols

Symbols generally represent memory addresses and may appear in both the location and the variable fields of the symbolic instructions. The programmer defines a symbol by placing it in the location field of an instruction, thus giving the instruction a symbolic address. The assembly program keeps track of the location of instructions in the source program by stepping a location counter by one for each instruction. When a symbol appears in the location field, it is normally assigned the current value of the location counter. The first such occurrence constitutes the definition of the symbol, and any subsequent occurrence in the location field will cause an error print-out. Undefined symbols, that is, symbols, appearing in the variable field of an instruction, and not in any location field, will cause an error print-out. The value of an undefined symbol is some location at the end of the program.

Symbols consist of 1 to 4 characters from among the 37-character set of the letters of the alphabet, the 10 digits and the dollar sign character (\$). At least one of the characters in any symbol must be alphabetic. The \$ character should be used with care since it is used in column 1 by the update program to flag a command card.

The following symbols are legitimate.

LOOP  
STP2

## Expressions

Expressions appear only in the variable field and may be either simple (composed of a single element) or compound (composed of two or more elements separated by operators). An element may be either a symbol, a decimal integer less than or equal to 65535, an octal integer preceded by an apostrophe less than or equal to '177777, a single asterisk, or a double asterisk.

When a single asterisk appears in the variable field as an element, it designates an address equal to the current value of the location counter. Thus, \* + 1 means "this location plus one." A double asterisk has a value of zero and is commonly placed in the variable field when the address is to be modified later by the program.

Operators are used to separate elements in compound expressions. An operator may be either a plus (addition), or a minus (subtraction). Only one operator is permissible between each pair of elements.

Expressions may have either relocatable or absolute modes. A relocatable expression is one that is relative to the first instruction of the program; an absolute expression is one which has a constant value regardless of its relative position in the program (e.g., an integer). The overall mode of the expression depends on the mode of each of the individual elements used to make up the expression.

Any permissible expression may be written to represent the address portion of a standard instruction. Additionally, the standard index (location zero) may be specified by following the address expression with a comma and the integer one.

The following are examples of valid expressions:

|                    |                                      |
|--------------------|--------------------------------------|
| Assume (P) is '203 | then Q + 5 = '12                     |
| Q = '5             | ZZ + 2 = '15                         |
| ZZ = '13           | * = '203                             |
| R = '20            | * - Q = '176                         |
|                    | * + 3 + Q - ** + '17303 - R = '17476 |

## Literals

Reference to a memory location containing a constant may be accomplished by use of one of the data defining pseudo-operations provided in the DAP-16 language. However, it is sometimes more convenient to represent a constant literally rather than symbolically. Consider the following example.

| PROGRAMMER |   |           |    | DATE       |    | PAGE     |
|------------|---|-----------|----|------------|----|----------|
| PROGRAM    |   |           |    |            |    | CHARGE   |
| LOCATION   | ① | OPERATION | ①  | ADDRESS. X | ①  | COMMENTS |
| 1          | 4 | 6         | 10 | 12         | 30 | 72 7     |
|            |   | LDA       |    | A          |    |          |
| A          |   | DEC       |    | 50         |    |          |

The first instruction refers to the symbolic constant A. The second instruction defines the constant as having the decimal value 50. An equivalent reference to the constant would have been as follows.

|            |  |   |           |  |      |            |  |  |    |          |  |    |   |  |
|------------|--|---|-----------|--|------|------------|--|--|----|----------|--|----|---|--|
| PROGRAMMER |  |   |           |  | DATE |            |  |  |    | PAGE     |  |    |   |  |
| PROGRAM    |  |   |           |  |      |            |  |  |    | CHARGE   |  |    |   |  |
| LOCATION   |  | ① | OPERATION |  | ①    | ADDRESS. X |  |  | ①  | COMMENTS |  |    | ① |  |
| 1          |  | 4 | 6         |  | 10   | 12         |  |  | 30 |          |  | 72 |   |  |
|            |  |   | LDA       |  |      | =50        |  |  |    |          |  |    |   |  |
|            |  |   |           |  |      |            |  |  |    |          |  |    |   |  |

In this example, DAP-16 interprets the =50 as a decimal literal, and automatically generates and assigns a location for the value 50. The resultant location of the value 50 is inserted into the address portion of the LDA instruction in the object program.

Three types of literals, decimal, octal, and ASCII, are interpreted by DAP-16. A decimal literal consists of the equals character (=), followed by the sign (if no sign, the number is positive), followed by a fixed-point decimal integer. The rules for forming an octal literal are identical, except that an apostrophe (') must follow the equals character. ASCII literals consist of the equals character followed by an A (=A), followed by two ASCII characters. If only one ASCII character is specified, the second character is assumed to be a blank. The ASCII literals are an exception to the rule governing blanks in the variable field. The two characters following the "A" form the literal and the third character must be either a blank (end of the variable field) or a comma (beginning of the index subfield).

#### Asterisk Conventions

The conventions for use of the asterisk are summarized below.

- An asterisk (\*) in column 1 or first character in the location field: treat the entire card or line as remarks.
- An asterisk (\*) appended to instruction mnemonic: set the indirect address flag.
- An asterisk(\*) as an element: current value of the location counter.
- A double asterisk (\*\*) as a symbolic address: put zeros in address field (address will be modified by another instruction).
- A triple asterisk (\*\*\*) as an operation code: op-code will be modified by another instruction. The instruction will be assembled as a memory reference instruction with an operation code of 00<sub>8</sub>.

#### ASSEMBLY LISTING

The printed output of DAP-16 is called the assembly listing. It is a printing of the symbolic input instructions in the order in which they appeared, together with the octal

representation of the binary words produced by the assembler. A sample listing is shown in Figure 2-1. The first column contains the line ID number, which identifies the line and is used by the source-program update routine. The next column shows the memory location assigned to each instruction. The third column shows, in octal, the binary word assigned to the location.

The following observations taken from Figure 2-1 are intended to aid the reader in analyzing the characteristics of DAP-16.

- a. Line 1 contains an asterisk in the location field, causing DAP-16 to treat the entire line as remarks.
- b. Line 2 contains a pseudo-operation (ORG) which sets the DAP-16 location counter to octal 1000, the starting address of sector one.
- c. The expression in the variable field in line 3 means the current value of the location counter, plus one. Consequently, DAP-16 has written octal 1001 into the address field of the instruction word assigned to this location.
- d. The symbol in the left margin of line 5 is a diagnostic. Diagnostics are explained in Section IV.
- e. In line 10, the programmer has entered the number of shifts desired in an LGL instruction. DAP-16 has generated the necessary two's complement form in the object program.
- f. Following line 10 is a literal pool of the three literals called for by the program.

### SECTION III DAP-16 PSEUDO-OPERATION

This section contains descriptions of all pseudo-operations provided in the DAP-16 language. Ancillary discussions of program relocation, data formatting, and program linkages are included to clarify pseudo-operation functions. For a description of machine language instructions, refer to the DDP-16 Programmers Reference Manual. For a summary listing of DAP-16 pseudo-operations, see Appendix A.

#### ASSEMBLY CONTROLLING PSEUDO-OPERATIONS

Assembly controlling pseudo-operations (ABS, CFx, END, FIN, LOAD, MOR, ORG, and REL) are used to start and stop program assembly, and to select the assembly mode. Programs may be assembled in either the absolute or relocatable mode. Relocatable programs can be placed anywhere within memory at the time of loading, whereas absolute programs must be placed in their assembled locations.

During program assembly, DAP-16 maintains a location counter to assign memory locations for each data and instruction word that is assembled. The output of the location counter is shown on the assembly listing (Figure 2-1). If the program is assembled in the absolute mode, the DAP-16 loader will load the object program into the locations shown on the assembly listing. If the program is assembled in the relocatable mode (specified by an REL pseudo-operation), the loader will load the object program into the memory area specified by the programmer at program loading time. It is recommended that the main program be loaded at a starting address equal to or greater than  $1000_8$ , so that sector zero can be used exclusively for address linkage and transfer vectors.

DAP-16, in the absence of a LOAD or REL pseudo-operation, assembles programs in the absolute mode. Relocatable programs are tentatively assembled for loading at a starting location of zero. However, at load time, a relocation constant is added to or subtracted from the address field of memory reference instructions and data words which reference symbolic locations. The relocation constant is equal to the difference between zero and the program starting location selected at load time.

When assembling relocatable programs, DAP-16 inserts control bits into the object program (not shown in the assembly listing) that enable the loader to identify instruction and data words referencing symbolic memory locations. The loader then adds the relocation constant to the address fields of these words.

### ABS Pseudo-Operation

The ABS (absolute) pseudo-operation is used to direct DAP-16 to assemble subsequent instructions in the absolute mode. The contents of the symbolic instructions containing the ABS pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | ABS     |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the ABS pseudo-operation is to assign absolute locations to the instructions assembled. The assembler then will continue to run in the absolute mode until a REL, LOAD or END pseudo-operation is encountered. The ABS mode is the normal assembly mode.

### CFx Pseudo-Operation

The CFx (configuration) pseudo-operation is used to inform DAP-16 as to which DDP-16 class computer the object program is to be executed on. The suffix "x" has the following connotation: 1 for the DDP-116, 4 for the DDP-416, and 5 for the DDP-516. If the configuration is not specified, it is assumed that the object program is to be executed on the same DDP-16 class computer as that on which the assembly is being performed. The contents of symbolic instructions containing the CFx pseudo-operation are as follows.

|           |                  |
|-----------|------------------|
| LOCATION  | Ignored          |
| OPERATION | CF1, CF4, or CF5 |
| VARIABLE  | Ignored          |
| COMMENTS  | Normal           |

The CFx pseudo-operation causes the DAP-16 to flag any instructions that are illegal for the object computer without interrupting the assembly.

### END Pseudo-Operation

The END pseudo-operation is used to direct DAP-16 to terminate the current assembly pass and prepare for the second pass if the two-pass mode has been selected. The contents of symbolic instructions containing the END pseudo-operation are:

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | END     |

|          |   |
|----------|---|
| VARIABLE | (1) An expression that defines the address of the instruction to which control should be transferred at the conclusion of the loading process at object time. If the variable field is left blank, the transfer address will be set to the location of the first instruction in the main program.<br>(2) Subroutine; ignored. |
| COMMENTS | Normal  |

The END pseudo-operation causes DAP-16 to perform the following functions:

- a. The current block of assembly output information is terminated.
- b. All literals are punched out and undefined symbols are assigned locations.
- c. An end jump block is punched following the assembly output. The jump address is the value of the expression in the variable field. If the variable field is left blank, the transfer address is set to the first instruction in the main program.
- d. The assembly process is terminated if the current pass is the final one.

The END pseudo-operation must be the last statement in the source program.

When operating in the two-pass mode, the START pushbutton must be depressed to start processing pass two. While the computer is halted, the operator must reposition the source tape to the beginning, or reload the card deck.

#### FIN Pseudo-Operation

The FIN (finish) pseudo-operation is used to direct DAP-16 to punch out all literals accumulated up to the point at which the FIN pseudo-operation is initiated. The contents of symbolic instructions containing the FIN pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | FIN     |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the FIN pseudo-operation is to cause DAP-16 to punch out all the accumulated literals. The purpose of this pseudo-operation is to permit literals to be interspersed throughout the program thus minimizing the necessity for indirect address links when referencing literals.

#### LOAD Pseudo-Operation

The LOAD pseudo-operation is used to direct DAP-16 to flag any instruction address that required desectorizing. The contents of symbolic instructions containing the LOAD pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | LOAD    |

|          |         |
|----------|---------|
| VARIABLE | Ignored |
| COMMENTS | Normal  |

The effect of the LOAD pseudo-operation is to cause DAP-16 to flag any instruction whose address refers to a location outside the current sector or sector zero. The assembler will continue to operate in the LOAD mode until an END, REL, or ABS pseudo-operation is encountered.

#### MOR Pseudo-Operation

The MOR (more) pseudo-operation causes the computer to halt and await operator action (except when magnetic tape input has been selected in which case MOR is ignored). The contents of symbolic instructions containing the MOR pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | MOR     |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

#### ORG Pseudo-Operation

The ORG (origin) pseudo-operation sets the location counter to a specified value. The contents of symbolic instructions containing the ORG pseudo-operation are as follows.

|           |  |
|-----------|--|
| LOCATION  | Normal   |
| OPERATION | ORG  |
| VARIABLE  | Normal. Any symbol used in this field must have been previously defined. |
| COMMENTS  | Normal   |

The ORG pseudo-operation performs the following functions.

- The expression in the variable field is evaluated.
- The location counter is set to the value thus determined.

A symbol in the location field of an ORG pseudo-operation is assigned the value of the location counter prior to processing the ORG pseudo-operation. Consider the following example.

| PROGRAMMER |   |   |           |    | DATE |            | PAGE   |
|------------|---|---|-----------|----|------|------------|--------|
| PROGRAM    |   |   |           |    |      |            | CHARGE |
| LOCATION   |   | ① | OPERATION |    | ②    | ADDRESS. X |        |
| ①          |   | ② |           | ③  |      | COMMENTS   |        |
| 1          | 4 | 6 | 10        | 12 | 30   | 72 73      |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |
|            |   |   |           |    |      |            |        |

The LDA instruction will be assigned to an absolute location (1000<sub>8</sub>). The symbol FUNA will be assigned the absolute value 101<sub>8</sub>.

#### REL Pseudo-Operation

The REL (relocatable) pseudo-operation is used to direct DAP-16 to assemble the subsequent instructions in the relocatable mode. The contents of symbolic instructions containing the REL pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | REL     |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the REL pseudo-operation is to cause DAP-16 to assign relative locations to the instructions assembled. The assembler will then continue to run in the relocatable mode until the END pseudo-operation is encountered or until an ABS or a LOAD pseudo-operation is encountered.

#### DATA DEFINING PSEUDO-OPERATIONS

The data defining pseudo-operations (BCI, DAC, DBP, DEC, and OCT) are used for defining constants and generating data for inclusion in the object program. The operations in this category cause DAP-16 to interpret alphanumeric data, decimal numbers, and octal numbers, respectively. The somewhat complex rules and restrictions for forming expressions in the variable field in the DEC pseudo-operation are discussed in the paragraphs immediately following the summary coverage of format and content.

Note that decimal and octal constants can also be generated by the use of literals, as discussed in Section II.

#### BCI Pseudo-Operation

The BCI (binary coded information) pseudo-operation is used to direct DAP-16 to generate binary words in ASCII form from alphanumeric data. The contents of symbolic instructions containing the BCI pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal  |
| OPERATION | BCI   |
| VARIABLE  | N, followed by 2N alphanumeric characters. The N specifies the number of words to be converted and may not exceed 29. |
| COMMENTS  | Normal  |

The effect of the BCI pseudo-operation is to convert each group of two characters into a left-justified binary word in ASCII code; these words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol

in the location field, it is assigned the same location as the first word of binary data generated by the pseudo-operation. The alphanumeric characters in the message to be encoded must be counted and entered as the first subfield. A typical example is shown below (six words of storage required). The BCI pseudo-operation is an exception to the rule in that the first blank terminates the variable field. The comments field begins immediately following the last character included in the character count.

|            |   |   |           |    |      |                 |  |  |  |        |          |  |  |    |   |
|------------|---|---|-----------|----|------|-----------------|--|--|--|--------|----------|--|--|----|---|
| PROGRAMMER |   |   |           |    | DATE |                 |  |  |  | PAGE   |          |  |  |    |   |
| PROGRAM    |   |   |           |    |      |                 |  |  |  | CHARGE |          |  |  |    |   |
| LOCATION   |   | ① | OPERATION |    | ①    | ADDRESS, X      |  |  |  | ①      | COMMENTS |  |  |    | ① |
| 1          | 4 |   | 6         | 10 |      | 12              |  |  |  |        | 30       |  |  | 72 |   |
| FINI       |   |   | BCI       |    |      | 6, REMOUNT TAPE |  |  |  |        |          |  |  |    |   |
|            |   |   |           |    |      |                 |  |  |  |        |          |  |  |    |   |

#### DAC Pseudo-Operation

The DAC (define address constant) pseudo-operation directs DAP-16 to generate a 16-bit binary word, which can be used by flagged memory reference instructions to access an operand in any memory sector. The contents of symbolic instructions containing the DAC pseudo-operation are as follows.

|           |             |
|-----------|-------------|
| LOCATION  | Normal      |
| OPERATION | DAC or DAC* |
| VARIABLE  | Normal      |
| COMMENTS  | Normal      |

The DAC pseudo-operation causes DAP-16 to evaluate the expression in the variable field and assemble a 16-bit address word. When the flag or tag (bit 1 or 2) is specified as part of the address word, the value of constant generated is increased by  $100000_8$  or  $40000_8$ , respectively. It is the programmer's responsibility to ensure that addresses over 37777 are not mistaken for flags and tags and vice-versa.

#### DEC Pseudo-Operation

The DEC (decimal) pseudo-operation is used to direct DAP-16 to generate binary words from decimal data. The contents of symbolic instructions containing the DEC pseudo-operation are as follows.

|           |        |
|-----------|--------|
| LOCATION  | Normal |
| OPERATION | DEC    |

|          |  |
|----------|--|
| VARIABLE | One or more subfields, each containing a decimal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters in the instruction line must not exceed 72. Rules for forming the decimal subfields are discussed below. |
| COMMENTS | Normal   |

The effect of the DEC pseudo-operation is to cause DAP-16 to convert each subfield to one, two, or three binary words, depending on whether the decimal data is single-precision fixed-point, double-precision fixed-point, single-precision floating-point or double-precision floating-point. These words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it is assigned the same location as the first word of binary data generated by the pseudo-operation.

Fixed-Point Decimal Data. -- Fixed-point decimal data may be either single precision or double precision. A significance of four decimal digits can be maintained in single-precision, fixed-point arithmetic on the DDP-16. In many arithmetic operations, this degree of significance is adequate and is desirable because of the enhanced speed of computation. A single-precision fixed-point decimal number requires one computer word (sign and 15 bits of significance) and is written in two parts: the significant part, and the scaling part. Double-precision fixed-point data consists of two words (sign and 30 significant bits).

The significant part of the fixed-point number is a signed or unsigned decimal number with or without a decimal point. If the decimal point is not specified, it is assumed to be immediately to the right of the last digit (a decimal integer).

The scaling part of the fixed-point number is the letter B (for single-precision) or the letters BB (for double-precision), followed by a signed or unsigned decimal integer specifying the position of the understood binary point. If the scaling part is not present, the number will be interpreted as a truncated decimal integer, whose understood binary point is immediately to the right of the least significant bit in the computer word (position 16).

The general form of the scaling part is  $B \pm NN$  or  $BB \pm NN$ , where NN gives the position of the understood binary point relative to the machine binary point. The minus sign defines the understood binary point to be to the left of the machine binary point, and the plus (or no sign) defines the understood binary point to be to the right of the machine binary point. The machine binary point is defined to be between the sign bit and the most significant bit of the computer word; i. e., between bit positions 1 and 2.

In addition to a scaling part, fixed-point numbers may also have an exponent part specified by the use of an E field in addition to a B field. E fields are discussed more fully in paragraphs on floating-point data.

The examples below show how DAP-16 produces fixed-point numbers. The left column shows the decimal number to be translated. This is written in the variable field. The right column shows the resultant octal word that would be generated by DAP-16. Single-precision, fixed-point numbers are limited to magnitudes less than  $2^{15}$ .

|           |        |
|-----------|--------|
| 15        | 000017 |
| 15B+15    | 000017 |
| 15.001B5  | 036001 |
| 15.001BB5 | 036001 |
|           | 003044 |
| -.002B-2  | 177372 |

Floating-Point Decimal Data. -- Floating-point data may be either single-precision or double-precision. A single-precision, floating-point number requires two computer words (sign, 8-bit characteristic, and 23-bit fraction). A double-precision, floating-point number requires three computer words (sign, 8-bit characteristic, and 39-bit fraction).

A decimal floating-point number is written as two parts: the significant part and the exponent part. The significant part of a floating-point number is a signed or unsigned decimal number written with a decimal point.

The exponent part of the decimal floating-point number is the letter E or the letters EE followed by a signed or unsigned decimal integer. The exponent part serves the following purposes.

- It indicates whether the floating-point number is to be single (E) or double-precision (EE).
- It specifies a constant in the form of 10 raised to the indicated power by which the significant part of the number is to be multiplied.

The resulting 8-bit binary exponent is expressed in 128 excess arithmetic and allows for numbers in the range  $10 \pm 2^{38}$ .

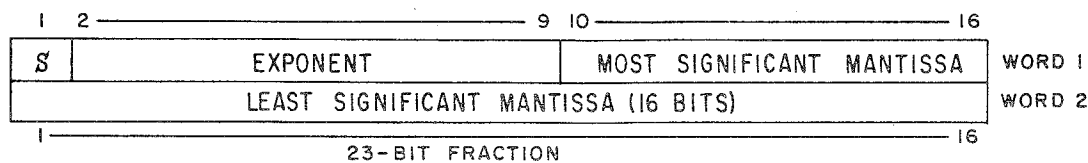
All negative floating-point numbers are expressed in two's complement form, which means that the exponent in this case is in one's complement form.

Figure 3-1 shows the formats of floating-point numbers and Table 3-1 shows various examples of floating-point numbers generated by the DEC pseudo-operation. The left-hand column shows the decimal number to be translated and the right-hand column shows the octal words that would be generated by the DEC pseudo-operation. The fractional portion of the floating-point number is always normalized by DAP-16.

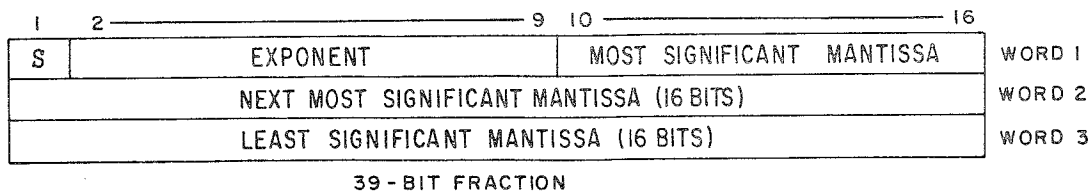
#### DBP Pseudo-Operation

The DBP (double precision) pseudo-operation directs DAP-16, when assembling on a DDP-516 with the double-precision option, to generate binary words from decimal data. The contents of symbolic instruction containing the DBP pseudo operation are as follows.

|           |        |
|-----------|--------|
| LOCATION  | Normal |
| OPERATION | DBP    |



A. Single-Precision Format



B. Double-Precision Format

Figure 3-1. Floating-Point Formats

Table 3-1.  
Floating-Point Number Translations

| Decimal Number | Octal Translation          | Remarks  |
|----------------|----------------------------|--|
| .15E2          | 041170<br>000000           | .15 times 10 <sup>2</sup> = 15                   |
| +.15E + 2      | 041170<br>000000           | Same as first example                            |
| -.15E2         | 136610<br>000000           | Negative of first example                        |
| 1234E-5        | 036545<br>013333           | Expression = .01234                              |
| .123           | 037375<br>171666           | Single-precision                                 |
| .1E0           | 037346<br>063146           | Single-precision; binary<br>exponent is negative |
| .1EE0          | 037346<br>063146<br>063146 | Double-precision result                          |

|          |   |
|----------|---|
| VARIABLE | One or more subfields, each containing a decimal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters in the instruction line must not exceed 72. |
|----------|---|

The effect of the DBP pseudo-operation is the same as that of the DEC pseudo-operation with the exception that the DBP always loads an even location and always generates a double-precision constant.

#### OCT Pseudo-Operation

The OCT (octal) pseudo-operation directs DAP-16 to generate binary words from octal data. The contents of symbolic instructions containing the OCT pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal  |
| OPERATION | OCT   |
| VARIABLE  | One or more subfields, each containing an octal data item. The subfields are separated by commas. The number of subfields is limited only by the restriction that the total number of characters on the instruction line must be limited to 72. |
| COMMENTS  | Normal  |

The effect of the OCT pseudo-operation is to cause DAP-16 to convert each subfield to a binary word. The octal data entries are right-justified, and assigned to successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it is assigned to the same location as the first word of binary data generated by the pseudo-operation.

The only allowable characters in an octal field are: plus, minus, apostrophe, 0, 1, 2, 3, 4, 5, 6, 7, and commas separating the subfields. Octal numbers may be signed (limited to magnitudes less than  $2^{15}$ ) or unsigned (limited to magnitudes less than  $2^{16}$ ). If an octal number is unsigned, it is assumed to be positive. The appearance of an apostrophe preceding the octal number is acceptable but is redundant.

#### LOADER-CONTROLLING PSEUDO-OPERATIONS

The loader-controlling pseudo-operations (EXD, LXD and SETB) are used to enter or leave the extended addressing mode for desectorizing and to designate a memory sector other than sector zero as the base sector for cross sector linkage. Pseudo-operations EXD and LXD are valid only for those DDP-516 computers equipped with the extended memory option. Pseudo-operation SETB is valid primarily for those DDP-516 computers equipped with the memory lockout option. Programs containing the EXD, LXD or SETB pseudo-operations must be loaded using the extended DAP/FORTRAN loader (LDR) rather than the standard loader (SLDR).

### EXD Pseudo-Operation

The EXD (enter extend-mode desectorizing) pseudo-operation directs the loader to desectorize the subsequent instructions for execution in the extended addressing mode. The contents of symbolic instructions containing the EXD pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | EXD     |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the EXD pseudo-operation is to increase the size of loader created indirect address words to 15 bits to increase addressing capability to 32K. This limits the extend mode to one level of indexing since the tag of the instruction word is not moved into the indirect address word. Therefore, bit 2 of the indirect address word is no longer interpreted as a tag but as part of the address.

### LXD Pseudo-Operation

The LXD (leave extend-mode desectorizing) pseudo-operation directs the loader to desectorize subsequent instructions for execution in the normal addressing mode. The contents of symbolic instructions containing the LXD pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | LXD     |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the LXD pseudo-operation is to restore loading to the normal addressing mode.

### SETB Pseudo-Operation

The SETB (set base sector) pseudo-operation notifies the loader that a base sector other than sector zero will be used to execute subsequent instructions. The contents of symbolic instructions containing the SETB pseudo-operation are as follows.

|           |  |
|-----------|--|
| LOCATION  | Normal   |
| OPERATION | SETB   |
| VARIABLE  | Normal. Any symbol used in this field must have previously been defined. |
| COMMENTS  | Normal   |

The pseudo-operation SETB designates the sector in which the indirect address words for cross sector linkage are to be stored. The value of the variable field designates the first location into which indirect address words are to be stored. Successive words are stored in successive locations. If a symbol appears in the location field, it will be assigned the current value of the location counter.

The SETB pseudo-operation does not reserve a block of storage for the indirect address word table. It is the programmer's responsibility to reserve a block for the table in the proper place via a BSS pseudo-operation.

#### LIST-CONTROLLING PSEUDO-OPERATIONS

The list-controlling pseudo-operations (EJCT, LIST, and NLST) are used to control the print-out of the source and object program assembly listing. These operations have no effect on the object program.

##### EJCT Pseudo-Operation

The EJCT (eject) pseudo-operation directs DAP-16 to begin or resume listing on a new page. The contents of symbolic instructions continuing the EJCT pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | EJCT    |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the EJCT pseudo-operation is to cause the I/O selector program (IOS) to generate the necessary commands to advance the listing one page and continue listing on a new page. This pseudo-operation is valid only with systems having a line printer and is ignored if the pseudo-operation NLST is currently in effect.

##### LIST Pseudo-Operation

The LIST (listing) pseudo-operation directs DAP-16 to print a side-by-side listing of the program being assembled. The contents of symbolic instructions containing the LIST pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | LIST    |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the LIST pseudo-operation is to cause the source program and its octal representation to be listed on the on-line typewriter or printer. The assembler then continues to operate in the listing mode until an NLST pseudo-operation is encountered. The assembler is normally in the LIST mode.

##### NLST Pseudo-Operation

The NLST (no listing) pseudo-operation directs DAP-16 to refrain from producing a side-by-side listing of the program being assembled. The contents of symbolic instructions containing the NLST pseudo-operation are as follows.

|           |         |
|-----------|---------|
| LOCATION  | Ignored |
| OPERATION | NLST    |
| VARIABLE  | Ignored |
| COMMENTS  | Normal  |

The effect of the NLST pseudo-operation is to inhibit DAP-16 from listing the source program and its octal representation on the on-line typewriter or printer. The assembler then continues to operate in the no-listing mode until a LIST pseudo-operation is encountered. Initialization of the assembler automatically sets the listing mode.

#### PROGRAM LINKING PSEUDO-OPERATIONS

The DAP-16 pseudo-operations CALL and SUBR are used to generate communication links between programs. The CALL pseudo-operation initiates transfer of control to an external subroutine. The SUBR pseudo-operation defines points of entry into the subroutine from an external program.

The variable field of the CALL pseudo-operation contains the name of the external subroutine being called. Each time a particular subroutine is called, DAP-16 punches the subroutine name as a special block and assembles a JST (jump and store) operation to location ZERO. Then, as the object program is loaded into memory, the loader completes the program linkage by requesting and loading the external subroutine being called, and filling in the address of the JST instruction, desectorizing it if necessary.

#### CALL Pseudo-Operation

The CALL (call) pseudo-operation directs DAP-16 to generate instructions that will transfer control to a specified subroutine. The contents of symbolic instructions containing the CALL pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal                                    |
| OPERATION | CALL                                      |
| VARIABLE  | A subroutine name (one to six characters) |
| COMMENTS  | Normal                                    |

The effects of the CALL pseudo-operation are as follows.

- a. The subroutine name from the variable field is punched as a special block type.
- b. A JST with an address of zero is entered into the sequence of assembled instructions.
- c. If there is a symbol in the location field, it is assigned to the location of the JST instruction inserted in step (b).

### XAC Pseudo-Operation

The XAC (external address constant) pseudo-operation directs the loader to generate a 16-bit binary word which is used by flagged memory reference instructions to access an operand outside the program. The contents of symbolic instructions containing the XAC pseudo-operation are as follows.

|           |  |
|-----------|--|
| LOCATION  | Normal   |
| OPERATION | XAC or XAC*  |
| VARIABLE  | External subroutine name (one to six characters),<br>optionally tagged |
| COMMENTS  | Normal   |

The XAC pseudo-operation causes the loader to evaluate the term in the variable field and assemble information which specifies that a reference is made outside the program. The external location must be defined either in the current or a separate program assembly by SUBR pseudo-operation. At load time, after the external reference is defined, the true address, the flag, and the tag are generated and stored at the location of the XAC word.

### SUBR Pseudo-Operation

The SUBR (subroutine) pseudo-operation is used to define a DAP-16 subroutine, and to symbolically assign a name to the subroutine for external reference.

The contents of symbolic instructions containing the SUBR pseudo-operation are as follows:

|           |  |
|-----------|--|
| LOCATION  | Ignored  |
| OPERATION | SUBR   |
| VARIABLE  | A one to six character name identifying an entry point to a subroutine optionally followed by a comma and a one to four character name defining the entry point. The name defining the entry point need be included only if it differs from the first four characters of the identifying name. |
| COMMENTS  | Normal   |

The effect of the SUBR pseudo-operation is to cause the identifying name in the variable field to be generated in the object program output as identification for the loader. There must be as many SUBR pseudo-operations in a subroutine as there are entry points; however, the entry points may be multiply defined. The SUBR pseudo-operation must be the first operation of the subroutine, preceded only by another SUBR, if present.

The following is an example of a subroutine for which entry and return provisions have been made.

| PROGRAMMER |   |           |    | DATE       |    | PAGE                   |
|------------|---|-----------|----|------------|----|------------------------|
| PROGRAM    |   |           |    |            |    | CHARGE                 |
| LOCATION   | ① | OPERATION | ①  | ADDRESS, X | ①  | COMMENTS               |
| 1          | 4 | 6         | 10 | 12         | 30 | 72                     |
|            |   | SUBR.     |    | SINE       |    |                        |
|            |   | REL.      |    |            |    |                        |
| SINE       |   | DAC       |    | * *        |    | START OF SINE ROUTINE  |
|            |   | JMP *     |    | SINE       |    | EXIT FROM SINE ROUTINE |

Access to this subroutine from an external program is possible by use of the following instruction.

| PROGRAMMER |   |           |    | DATE       |    | PAGE     |
|------------|---|-----------|----|------------|----|----------|
| PROGRAM    |   |           |    |            |    | CHARGE   |
| LOCATION   | ① | OPERATION | ①  | ADDRESS, X | ①  | COMMENTS |
| 1          | 4 | 6         | 10 | 12         | 30 | 72 73    |
|            |   | CALL      |    | SINE       |    |          |

The following subroutine has two entry points, and each entry point is defined twice.

| PROGRAMMER |   |           |    | DATE         |    |                                 |   | PAGE   |   |  |  |
|------------|---|-----------|----|--------------|----|---------------------------------|---|--------|---|--|--|
| PROGRAM    |   |           |    |              |    |                                 |   | CHARGE |   |  |  |
| LOCATION   | ① | OPERATION | ①  | ADDRESS, X   | ①  | COMMENTS                        | ① |        |   |  |  |
| 1          | 4 | 6         | 10 | 12           | 30 |                                 |   | 72     | 7 |  |  |
|            |   | SUBR      |    | SINE         |    | NAME FOR SINE ROUTINE           |   |        |   |  |  |
|            |   | SUBR      |    | COSINE       |    | NAME FOR COSINE ROUTINE         |   |        |   |  |  |
|            |   | SUBR      |    | ARCTAN, ATAN |    | NAME FOR ARCTAN ROUTINE         |   |        |   |  |  |
|            |   | SUBR      |    | SINF, SINE   |    | ALTERNATE NAME FOR SINE ROUTINE |   |        |   |  |  |
|            |   | REL       |    |              |    |                                 |   |        |   |  |  |
| SINE       |   | DAC       |    | * *          |    | START OF SINE ROUTINE           |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |
|            |   | JMP*      |    | SINE         |    | EXIT FROM SINE ROUTINE          |   |        |   |  |  |
| COSI       |   | DAC       |    | * *          |    | START OF COSINE ROUTINE         |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |
|            |   | JMP*      |    | COSI         |    | EXIT FROM COSINE ROUTINE        |   |        |   |  |  |
| ATAN       |   | DAC       |    | * *          |    | START OF ARCTAN ROUTINE         |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |
|            |   | JMP*      |    | ATAN         |    | EXIT FROM ARCTAN ROUTINE        |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |
|            |   |           |    |              |    |                                 |   |        |   |  |  |

Entry to the sine portion of the subroutine is made by

CALL SINE

or

CALL SINF

Entry to the cosine portion of the subroutine is made by

CALL COSINE

Entry to the arc tangent portion of the subroutine is made by

CALL ARCTAN

Programs coded as subroutines (i.e., programs preceded by the SUBR pseudo-operation) cannot be loaded independently by means of the DAP-16 loader, but must be called by a main program.

### STORAGE ALLOCATION PSEUDO-OPERATIONS

The DAP-16 pseudo-operations (BES, BSS, BSZ, and COMN) enable the programmer to allocate memory cells for data storage or working space. For example, if a group of 350 integers are to be ordered and assembled in a table, the symbolic instruction shown below allocates 350 consecutive cells for storage of the integers, in symbolic locations TABL through TABL + 349.

| PROGRAMMER |   |           |    | DATE       |    | PAGE     | C     |
|------------|---|-----------|----|------------|----|----------|-------|
| PROGRAM    |   |           |    |            |    | CHARGE   |       |
| LOCATION   | ① | OPERATION | ①  | ADDRESS. X | ①  | COMMENTS | ②     |
| 1          | 4 | 6         | 10 | 12         | 30 |          | 72 73 |
| TABL       |   | BSS       |    | 350        |    |          |       |

### BES Pseudo-Operation

The BES (block ending with symbol) pseudo-operation is used for reserving storage locations. The contents of symbolic instructions containing the BES pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal  |
| OPERATION | BES   |
| VARIABLE  | Any absolute expression. Any symbol used in this field must have been previously defined. |
| COMMENTS  | Normal  |

The effect of the BES pseudo-operation is to increase the value of the location counter by the value of the expression in the variable field. If there is a symbol in the location field, it is assigned the value of the location counter after the increase. Consider the following example.

| PROGRAMMER |   |           |    | DATE       |    | PAGE     | C     |
|------------|---|-----------|----|------------|----|----------|-------|
| PROGRAM    |   |           |    |            |    | CHARGE   |       |
| LOCATION   | ① | OPERATION | ①  | ADDRESS. X | ①  | COMMENTS | ②     |
| 1          | 4 | 6         | 10 | 12         | 30 |          | 72 73 |
| A          |   | ØCT       |    | 5          |    |          |       |
| BLK        |   | BES       |    | 5          |    |          |       |
| B          |   | ØCT       |    | 6          |    |          |       |

If A has been assigned location 50, BLK will be assigned location 56, leaving five vacant cells; B will also be assigned to location 56.

### BSS Pseudo-Operation

The BSS (block starting with symbol) pseudo-operation is used for reserving storage locations. The contents of symbolic instructions containing the BSS pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal  |
| OPERATION | BSS   |
| VARIABLE  | Any absolute expression. Any symbol used in this field must have been previously defined. |
| COMMENTS  | Normal  |

The effect of the BSS pseudo-operation is to increase the value of the location counter by the value of the expression in the variable field. If there is a symbol in the location field, it is assigned the value of the location counter before the increase. Consider the following example.

| PROGRAMMER |   |     |            | DATE |          | PAGE   |
|------------|---|-----|------------|------|----------|--------|
| PROGRAM    |   |     |            |      |          | CHARGE |
| LOCATION   | ① | ①   | ADDRESS, X | ①    | COMMENTS | ②      |
| 1          | 4 | 6   | 10         | 12   | 30       | 72 7   |
| A          |   | 0CT | 5          |      |          |        |
| BLK        |   | BSS | 5          |      |          |        |
| B          |   | 0CT | 6          |      |          |        |

In this case, if A has been assigned location 50, BLK will be assigned location 51, and B will be assigned location 56, leaving five vacant cells.

The BES and BSS pseudo-operations effect the punched output during assembly. When DAP-16 encounters one of these pseudo-operations, the block of machine instructions being accumulated in a special punch buffer (internal to DAP-16) is punched out, regardless of the number of words that have been accumulated. For BES and BSS, a new block is started with an origin address equal to the DAP-16 location counter after processing the BES or BSS pseudo-operation.

### BSZ Pseudo-Operation

The BSZ (block storage of zeros) pseudo-operation is used for reserving storage locations that are initially (at load time) set to ZEROs. The contents of symbolic instructions containing the BSZ pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal  |
| OPERATION | BSZ   |
| VARIABLE  | Any absolute expression. Any symbol used in this field must have been previously defined. |
| COMMENTS  | Normal  |

The effect of the BSZ pseudo-operation is to increase the value of the location counter by the value of the expression in the variable field. If there is a symbol in the location field, it is assigned the value of the location counter before the increase.

### COMN Pseudo-Operation

The COMN (common) pseudo-operation is used for assigning absolute storage locations in upper memory. The contents of symbolic instructions containing the COMN pseudo-operation are as follows.

|           |   |
|-----------|---|
| LOCATION  | Normal  |
| OPERATION | COMN  |
| VARIABLE  | Any absolute expression. Any symbol used in this field must have been previously defined. |
| COMMENTS  | Normal  |

The effect of the COMN pseudo-operation is to cause DAP-16 to subtract the value of the expression in the variable field from the COMMON base and assign this value to the symbol in the location field. COMMON base is a user option. The COMN pseudo-operation establishes a common data pool that can be referenced by several programs.

### SYMBOL DEFINING PSEUDO-OPERATION

A symbol defining pseudo-operation (EQU) is provided for assigning an absolute or relocatable value to a symbol.

### EQU Pseudo-Operation

The EQU (equals) pseudo-operation is used for defining a value for a symbol for reference by other DAP-16 operations. The contents of symbolic instructions containing EQU pseudo-operation are as follows.

|           |  |
|-----------|--|
| LOCATION  | Normal; must contain a symbol  |
| OPERATION | EQU  |
| VARIABLE  | Any absolute or relocatable expression. Any symbol used in this field must have been previously defined. |
| COMMENTS  | Normal   |

The EQU pseudo-operation causes DAP-16 to evaluate the variable field expression for value, and to assign the value to the symbol in the location field. The mode of the symbol in the location field will be the same as the mode of the expression in the variable field.

#### SPECIAL MNEMONIC CODES

Two special mnemonic codes are provided for the convenience of the programmer when writing special instruction groups for calling sequences. The mnemonic codes are assembled like any machine language instruction, in that they may have address, index, and indirect fields. These codes are desectorized by the loader as 9-bit address memory reference instructions.

##### Mnemonic

PZE

\*\*\*

##### Assembles As

ZEROs in op-code

ZEROs in op-code

## SECTION IV DAP-16 OPERATING INSTRUCTIONS

This section contains instructions for preparing the source program and for using DAP-16 to obtain an object program from the source program. A discussion of diagnostic print-outs provided by DAP-16 as an aid in debugging the source program, and a description of the object program is included.

### SOURCE PROGRAM PREPARATION

A DAP-16 source program is prepared on the coding form described in Section II. It consists of a set of symbolic instructions terminated by an END pseudo-operation. (The pseudo-operation MOR may be substituted under certain conditions.) The source program is then punched on paper tape or cards.

#### Paper Tape

To increase paper tape efficiency as an input medium to DAP-16, a delineating code is used to define the separation between fields. For example, if the location field is not used, it is not necessary to space five times to be in position for the operation field. A reverse slash character (\) is used as the delineating code on the ASR-33 or ASR-35. In addition, the carriage return, followed by a line feed, will terminate the entire line. The general format for the line follows.

LINE FEED, location field, reverse slash (or spaces to bring total number of characters and space in location field to five), operation field, reverse slash (or spaces to bring total number of characters and spaces in operation field to six), variable field, reverse slash (or one or more spaces), comments field, X-OFF, CARRIAGE RETURN. (The ASR-35 requires an X-OFF, CARRIAGE RETURN and RUBOUT at the end of a line because it reads two characters following an X-OFF, whereas the ASR-33 reads only one character.

The source program is terminated by the END pseudo-operation.

#### Cards

When using cards, no purpose is served by trying to make a line of code more compact, since the entire card must be read. Therefore, the card columns are used to define the fields. The only exception to this is the termination of the variable field and the

start of the comments field. DAP-16 will assume that the comments field starts after the first blank column following the variable field. If a blank is embedded within the variable field, DAP-16 will assume the remainder of the line to be comments (except in the BCI variable field or for ASCII literals). The general format for the card is as follows.

|                         |   |
|-------------------------|---|
| LOCATION FIELD          | Columns 1 to 4                                |
| OPERATION FIELD         | Columns 6 to 10                               |
| VARIABLE FIELD          | Columns 12 to first blank column or column 72 |
| COMMENTS FIELD          | First blank column to column 72               |
| IDENTIFICATION<br>FIELD | Columns 73 to 80                              |

When preparing cards, symbols must be left-justified in their respective fields. Columns 73-80 on the card are ignored by the assembly program and may be used to sequence the card deck.

#### OBJECT PROGRAM PREPARATION

Object program preparation consists of reading DAP-16 into computer memory, then reading the source tape or card deck, with the contents of the A-register set to provide the desired punching and printing options. Table 4-1 shows the significance of the various bit positions on both standard systems and those systems equipped with standard options. Principal options provided by DAP-16 are as follows.

- a. Punching the object program
- b. Punching or printing the assembly listing
- c. Punching the object program and printing the assembly listing simultaneously
- d. Assembling multi-section programs

For very brief programs, option (c.) provides an assembly listing for reference and simultaneously, an object program for execution. When an assembly listing is desired for programs of normal length, and a high-speed paper tape punch is available, the option of punching the assembly listing is most useful. The printed assembly listing can then be prepared off-line. Option (d.) is useful for assembling programs prepared in several sections by use of the MOR pseudo-operation.

Table 4-1.  
A-Register Bit Settings For I/O Device Selection

| Bit | Meaning  | Selection     |
|-----|--|---------------|
| 2   | Teletype   | Source Device |
| 3   | Paper Tape Reader                                      |               |
| 4   | Card Reader  |               |
| 5   | Magnetic Tape No. 1                                    |               |
| 6   | Teletype with program halts provided for manual inputs |               |

Table 4-1. (Cont)  
A-Register Bit Settings For I/O Device Selection

| Bit | Meaning             | Selection     |
|-----|---------------------|---------------|
| 7   | Teletype            | Object Device |
| 8   | Paper Tape Punch    |               |
| 9   | Card Punch          |               |
| 10  | Magnetic Tape No. 2 |               |
| 11  | No Object Output    |               |
| 12  | Teletype            | List Device   |
| 13  | Paper Tape Punch    |               |
| 14  | Magnetic Tape No. 2 |               |
| 15  | Line Printer        |               |
| 16  | No Listing          |               |

### ERROR DIAGNOSIS

DAP-16 is able to detect many types of clerical errors commonly made in coding programs. These errors are indicated by an appropriate error code printed in the left margin of the assembly listing (refer to Figure 2-1). Examples of errors that are detected and their associated flags are as follows.

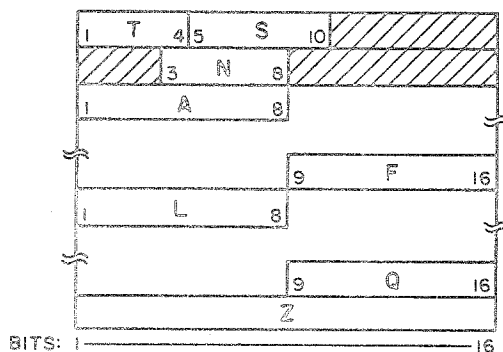
| <u>Error</u>   | <u>Flag</u> |
|--|-------------|
| Multiply defined symbol  | M           |
| Erroneous conversion of a constant or a variable field in improper format  | C           |
| Address field missing where normally required, or error in address format  | A           |
| Operation code missing or in error   | O           |
| Location symbol missing where required, or error in location symbol  | L           |
| Address of variable field expression not in sector being processed or sector zero (applicable only in load mode) | S           |
| Relocation assignment error  | R           |
| Symbol table or literal table exceeded   | X           |
| Major formatting error   | F           |
| Unclassified error in variable field of multiple field pseudo-operator (i. e., DEC, OCT, etc.)                   | V           |
| Improper use of or error in index field  | T           |

Errors in a field will generally result in that field being assembled as a 0. In the case of multiply defined symbols, the first symbol definition is used. If the operation code is illegal for computer configuration, the assembly is performed and the illegal codes are flagged with an "O".

## OBJECT PROGRAM FORMAT

The object is used by DAP-16 when assembling programs in the DESECTOR-IZING mode. This mode allows for relocatable main programs and subroutines in addition to absolute programs. Data are outputted in blocks composed of a parameter byte, followed by a data-word byte, then a logical difference checksum. There are eight block types (0-7) which are identified by bits 1 through 4 of the first word in the block. Block type zero is further subdivided into subblocks which are identified by bits 5 through 10 of the first word in the block. The following paragraphs contain a description of the various block types and their format.

### Block Type 0-0 Subprogram Name



T = 0, the block type

S = 0, the subblock type

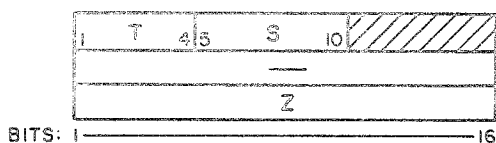
N is number of 16-bit words in the block including the checksum and control words

A-F is six-character name of the first entry point into the subprogram

L-Q is six-character name of the last entry point into the subprogram in this block.

Z is checksum for all words in block except for the checksum word

### Block Types 0-1, 0-2, and 0-3 Special Action



T = 0

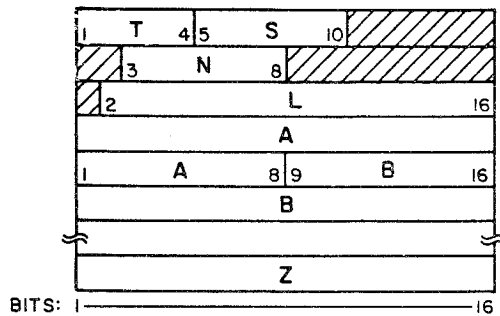
S = 1, turn off non-load flag

S = 2, turn on chain flag

S = 3, end-of-job

Z is checksum

## Block Type 0-4 Data



T = 0

S = 4

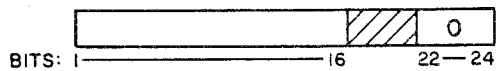
N is number of 16-bit words in the block

L is 15-bit address of location into which the first data is to be loaded. Successive words are loaded into location L + 1, L + 2, etc.

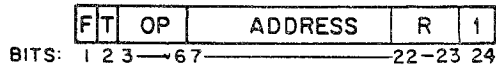
A, B... are data words in 24-bit format

Z is checksum

The data-word bytes have several formats, depending upon the last three bits of the byte. These formats are as follows.



Unmodified data generic or shift

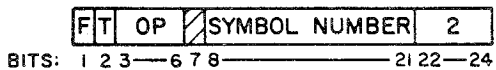


Address is known and to be desectorized

R = 0, absolute

R = 1, positively relocatable

R = 3, negatively relocatable

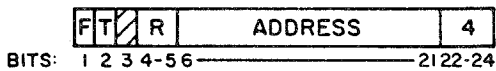


Symbolic address, to be desectorized when the address is known

Bit 8 = 0, this is the last symbol number associated with the address

Bit 8 = 1, the following symbol number is

also associated with the address. The following symbol number will appear in bits 8-21 of the next data word providing the current word is not the last word in the current data block. If the current word is the last word in the current block, the symbol number will appear in the next data block.

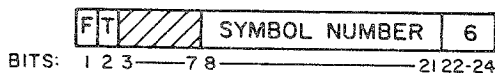


Address is known, do not desectorize

R = 0, absolute

R = 1, positively relocatable

R = 3, negatively relocatable

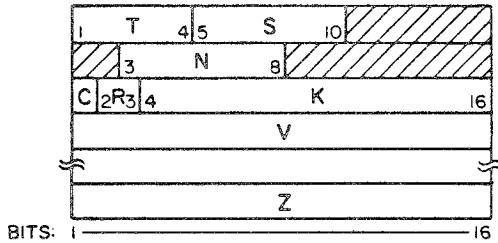


Symbolic address, not to be desectorized when the address is known.

Bit 8 = 0, this is the last symbol number associated with the address.

Bit 8 = 1, the following symbol number is also associated with the address. The symbol number may appear in the next block if the current word is the last word in the current data block.

#### Block Type 0-10 Symbol Number Definition Block



T = 0

S = 10

C = 0, symbol is referred to only once

C = 1, symbol is referred to more than once

R = 0, absolute

R = 1, positively relocatable

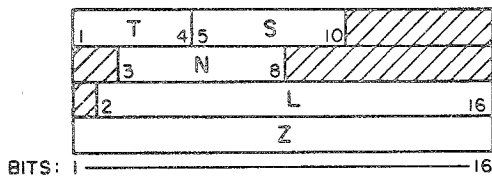
R = 3, negatively relocatable

K is 13-bit symbol number

V is 16-bit symbol value (positive or negative)

Z is checksum

#### Block Type 0-14 End



T = 0

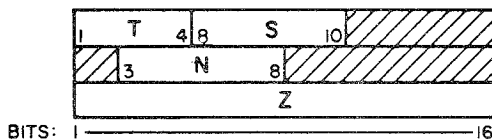
S = 14

N is number of 16-bit words in the block (always 4)

L is the jump address if this is the end of a main program. L is zero if this is the end of a subprogram

Z is checksum

#### Block Types 0-24, 30, 54, 60 Modes



T = 0

S = 24, relocatable mode

S = 30, absolute mode

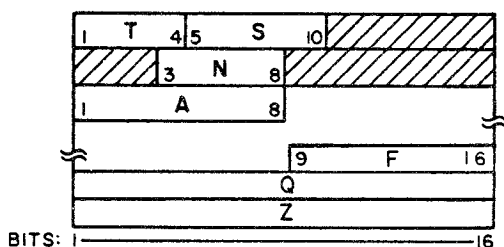
S = 54, enter extended-memory desectorizing mode

S = 60, leave extended-memory desectorizing mode

N is number of 16-bit words in the block (always 3)

Z is checksum

# Block Type 0-44 Subprogram Call



T = 0

S = 44

N is number of 16-bit words in the block  
(always 7)

A-F is six-character name of the entry point

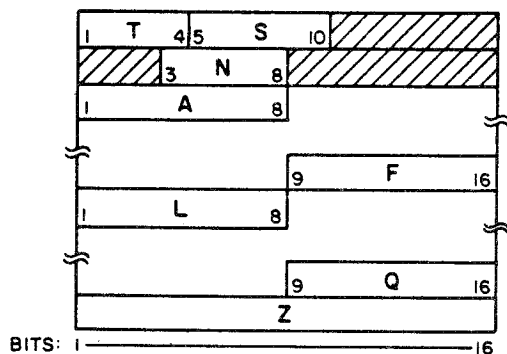
Q = 1, reference is not to be desectorized

Q = 0, reference is to be desectorized

Z is checksum

The last data word loaded is a reference to  
this subroutine name.

# Block Type 0-50 Subprogram Entry Point Definition



T = 0

S = 50

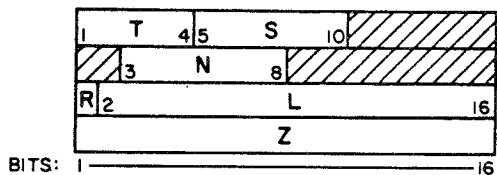
N is number of 16-bit words in the block

A-F is the first six-character name of this  
entry point into the subprogram

L-Q is the last six-character name of this  
entry point into the subprogram

Z is checksum

# Block Type 0-64 Set Base Sector



T = 0

S = 64

N is number of 16-bit words in the block (always 4)

R = 0, absolute location

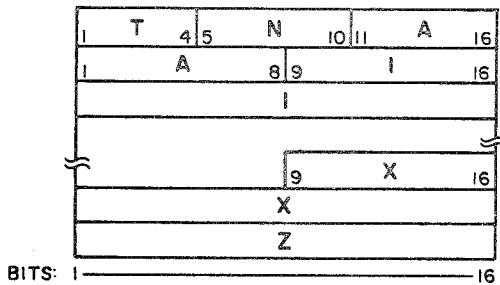
R = 1, relocatable location

L is 15-bit address of location at which the cross-  
sector indirect word table begins

Z is checksum

Block Types 0-20, 0-34, and 0-40 are illegal. They are reserved for internal functions of DAP-16.

#### Block Types 1 and 2 Program Words



T = 1, absolute program words

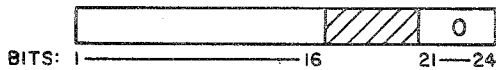
T = 2, relative program words

N is number of 16-bit words in the block

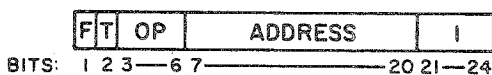
I-X is 24-bit data words

Z is checksum

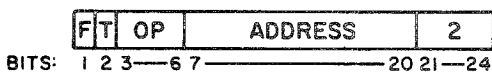
The data-word bytes in this block have several formats depending upon the last four bits of the byte. These formats are shown as follows.



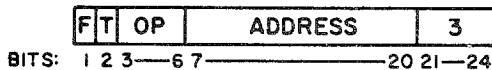
Load first 16 bits into memory unchanged



Address is not altered

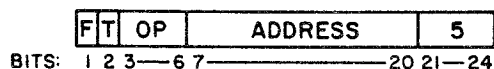


Address is positively relocated (add  $\Delta$ )

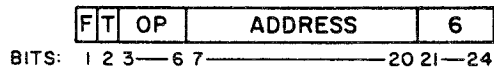


Address is negatively relocated (add  $\Delta$ , complement)

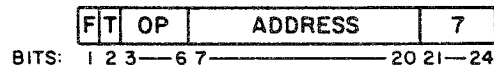
For types one through three, the address modified is interpreted as a 9-bit quantity. In case of an intersector reference, an indirect reference to sector zero is created.



Address is not altered



Address is positively relocated



Address is negatively relocated

For types 5 through 7, the resultant address is merely combined with the F and T fields before loading.

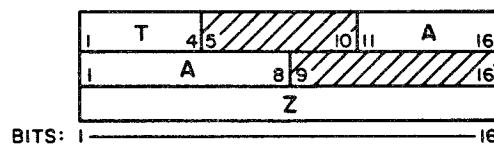


U is all ONEs

V is relative address of an instruction in a string of instructions each of which uses the same symbol. The relative address (V) is supplied to each instruction requiring the address. The string may contain DAC pseudo-operations

which accept a full 14-bit address and are distinguished from those instructions requiring a 9-bit address by their zero operation code. No instruction in a string may be desectorized into a base sector other than the currently active base sector.

#### Block Types 3 and 4 End Jumps



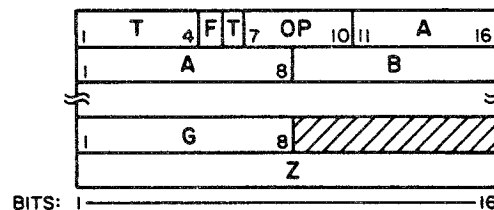
T = 3, jump address is absolute

T = 4, jump address is relative

A is jump address

Z is checksum

#### Block Types 5 and 7 Subroutine or Reference Call



T = 5, Subroutine call

T = 6, reference to an item in common

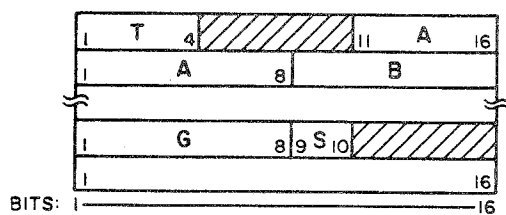
A is address of instruction. If T = 5 the address is relative to the common base sector

B-G is six-character name of subroutine or common item

Z is checksum

If C = 5, the operation code is JST\*

# Block Type 6 Subroutine or Common Block Definition



T = 6

A is entry point relative to the beginning of the subroutine if S = 0 or 2

= size of the common block if S = 1 or 3

B-G is six-character name of subroutine or common block

S = 0, subroutine definition

S = 1, common block definition

S = 2, subroutine definition

S = 3, data storage in common follows this block

Z is checksum

## SECTION V PROGRAMMING EXAMPLES

This section of the manual contains a program example. It is not intended to be executable but is given to illustrate various DAP-16 pseudo-operation features and error flags. Refer to the appropriate DDP-16 Programmers Reference Manual for additional programming examples.

|      |        |                 |   |                             |      |
|------|--------|-----------------|---|-----------------------------|------|
| 0001 |        |                 | * C500-001-6504 (DAP-TEST) CONTROL NUMBER 7011657 | REV. A                      | 0010 |
| 0002 |        |                 | * START OBJECT PROGRAM AT OCTAL 10                |                             | 0020 |
| 0003 |        |                 | *   |                             | 0030 |
| 0004 |        |                 | * PROGRAM SHOULD TYPE 'O.K.' AND HALT             |                             | 0040 |
| 0005 |        |                 | *   |                             | 0050 |
| 0006 |        |                 | *   |                             | 0060 |
| 0007 |        |                 | ORG *210  |                             | 0070 |
| 0008 | 00210  | 0 02 00274      | LDA 00  | COMPUTE CHECKSUM            | 0080 |
| 0009 | 00211  | 0 04 00000      | STA 0   | *                           | 0090 |
| 0010 | 00212  | 140040          | CRA   | *                           | 0100 |
| 0011 | 00213  | 1 05 00276      | ERA TT+1.1  | *                           | 0110 |
| 0012 | 00214  | 0 12 00000      | IRS 0   | *                           | 0120 |
| 0013 | 00215  | 0 01 00213      | JMP *-2   | *                           | 0130 |
| 0014 | 00216  | 0 11 00277      | CAS CKSM  | *                           | 0140 |
| 0015 | 00217  | 000000          | HLT   | *                           | 0150 |
| 0016 | 00220  | 0 01 00222      | JMP **2   | *                           | 0160 |
| 0017 | 00221  | 000000          | HLT   | WRONG SUM                   | 0170 |
| 0018 | 00222  | 0 02 ** T       | LDA =-3   | RIGHT SUM                   | 0180 |
| 0019 | 00223  | 0 04 00000      | SIA 0   | TYPE 'O.K.'                 | 0190 |
| 0020 | 00224  | 1 02 00245 TTTT | LDA MSG+3.1                                       |                             | 0200 |
| 0021 | 00225  | 07 0104         | SKS *104  | *                           | 0210 |
| 0022 | 00226  | 0 01 00225      | JMP *-1   | *                           | 0220 |
| 0023 | 00227  | 03 0104         | OCP *104  | *                           | 0230 |
| 0024 | 00230  | 0406 /0         | ARR 8   | *                           | 0240 |
| 0025 | 00231  | 17 0004         | OTA 4   | *                           | 0250 |
| 0026 | 00232  | 0 01 00231      | JMP *-1   | *                           | 0260 |
| 0027 | 00233  | 0416 /0         | ALR 8   | *                           | 0270 |
| 0028 | 00234  | 17 0004         | OTA 4   | *                           | 0280 |
| 0029 | 00235  | 0 01 00234      | JMP *-1   | *                           | 0290 |
| 0030 | 00236  | 0 12 00000      | IRS 0   | *                           | 0300 |
| 0031 | 00237  | 0 01 00224      | JMP TTTT  | *                           | 0310 |
| 0032 | 00240  | 000000          | HLT   | TEST COMPLETE               | 0320 |
| 0033 | 00241  | 177775          | FIN   |                             | 0330 |
| 0034 | 00242  | 106612          | MSG OCT 106612                                    | CARRIAGE RETURN + LINE FEED | 0340 |
| 0035 | 00243  | 147656          | BCI 2=0.K.  |                             | 0350 |
|      | 00244  | 145656          |   |                             |      |
| 0036 |        |                 | *   |                             | 0360 |
| 0037 |        |                 | * THE FOLLOWING CHECKS DAP OPERATION              |                             | 0370 |
| 0038 | 00245  | 0 02 00000 XX   | LDA +1-1+2-2+3-3                                  |                             | 0380 |
| 0039 |        | 177534 YY       | EQU -XX+1   |                             | 0390 |
| 0040 |        | 000011 ZZ       | EQU 3+3+3   |                             | 0400 |
| 0041 |        | 000145 M        | EQU XX-*100                                       |                             | 0410 |
| 0042 | 00246  | 0 177356        | DAC YY-ZZ-M                                       |                             | 0420 |
| 0043 | 00247  | -1 00 00250     | PZE* XX+3.1                                       |                             | 0430 |
| 0044 | 00250  | 120240          | BCI 2.  |                             | 0440 |
|      | 00251  | 120240          |   |                             |      |
| 0045 | 00252  | 000001          | OCT 1.3.144.-4.77777                              |                             | 0450 |
|      | 00253  | 000003          |   |                             |      |
|      | 00254  | 000144          |   |                             |      |
|      | 00255  | 177774          |   |                             |      |
|      | 00256  | 077777          |   |                             |      |
| 0046 | 00257  | 071143          | DEC .99E+30                                       |                             | 0460 |
|      | 00260  | 173346          |   |                             |      |
| 0047 | 00261  | 007320          | DEC .99E-30                                       |                             | 0470 |
|      | 00262  | 050576          |   |                             |      |
| 0048 | 00263  | 177372          | DEC -.0028-2.15.0018+5.15.00188+5                 |                             | 0480 |
|      | 00264  | 036001          |   |                             |      |
|      | 00265  | 036001          |   |                             |      |
|      | 00266  | 001422          |   |                             |      |
| 0049 | 00267  | 036545          | DEC 1234.E-5..15FE-2                              |                             | 0490 |
|      | 00270  | 013333          |   |                             |      |
|      | 00271  | 035742          |   |                             |      |
|      | 00272  | 046722          |   |                             |      |
|      | 00273  | 170651          |   |                             |      |
| 0050 | 00274  | -1 177724 00    | DAC* -TT+T-1.1                                    |                             | 0500 |
| 0051 | 00275  | -1 00 00250 TT  | PZE* XX+3.1                                       |                             | 0510 |
| 0052 | 00276  | 0 000000        | DAC **  |                             | 0520 |
| 0053 | 00277  | 105314          | CKSM OCT 105314                                   |                             | 0530 |
| 0054 | 00312  |                 | BES 10  |                             | 0540 |
| 0055 | 00312  |                 | BSS 10  |                             | 0550 |
| 0056 | 000324 | LIM EQU *       |   |                             | 0560 |
| 0057 |        | END             |   |                             | 0570 |

# APPENDIX A SUMMARY OF DAP PSEUDO-OPERATIONS

| Oper.<br>Mnemonic | Meaning                            | Class                       | Contents of Fields   |                               |   | Effect   |
|-------------------|------------------------------------|-----------------------------|--|-------------------------------|---|--|
|                   |                                    |                             | Location   | Operation                     | Variable  |  |
| ***               | ZERO<br>op-code                    | Special<br>mnemonic<br>code | Normal   | ***                           | Normal  | ZEROs put into op-code   |
| ABS               | Absolute<br>mode                   | Assembly<br>control         | Not Applicable   | ABS                           | Ignored   | Subsequent instructions<br>assembled in absolute<br>mode   |
| BCI               | Binary<br>coded in-<br>formation   | Data de-<br>fining          | Normal   | BCI                           | N, followed by 2N<br>alphanumeric<br>characters                       | 2N alphanumeric char-<br>acters (N<30) converted<br>into binary  |
| BES               | Block end-<br>ing with<br>symbol   | Storage<br>allocation       | Normal; assigned<br>location counter<br>value after<br>increase    | BES                           | Previously defined<br>absolute expres-<br>sion                        | Increases value of loca-<br>tion counter by value of<br>expression in the vari-<br>able field  |
| BSS               | Block start-<br>ing with<br>symbol | Storage<br>allocation       | Normal; assigned<br>location counter<br>value before in-<br>crease | BSS                           | Previously defined<br>absolute expres-<br>sion                        | Same as BES  |
| BSZ               | Block stor-<br>age of<br>ZEROs     | Storage<br>allocation       | Normal   | BSZ                           | Previously defined<br>absolute expres-<br>sion                        | Sames as BES (used for<br>defining storage blocks<br>that are initially cleared)   |
| CALL              | Call sub-<br>routine               | Program<br>linking          | Normal   | CALL                          | Name of a subrou-<br>tine   | Generates a JST* to call<br>referenced subroutine<br>through transfer vector   |
| CFx               | Configura-<br>tion                 | Assembly<br>control         | Not Applicable   | CF1<br>or<br>CF4<br>or<br>CF5 | Ignored   | Specifies which DAP-16<br>class computer will ex-<br>ecute the object program.<br><br>CF1 for DDP-116<br>CF4 for DDP-416<br>CF5 for DDP-516  |
| COMN              | Put in com-<br>mon stor-<br>age    | Storage<br>allocation       | Normal   | COMN                          | Previously defined<br>absolute expres-<br>sion                        | Value of expression in<br>variable field is used to<br>assign location in a com-<br>mon data pool for symbol<br>in location field; facili-<br>tates reference by other<br>programs |
| DAC               | Define ad-<br>dress con-<br>stant  | Data de-<br>fining          | Normal   | DAC                           | Previously defined<br>absolute or relo-<br>catable expression         | Causes DAP-16 to assem-<br>ble a 16-bit address word   |
| DBP               | Double pre-<br>cision              | Data de-<br>fining          | Normal   | DBP                           | Decimal subfields   | Decimal characters con-<br>verted into binary with<br>double precision option  |
| DEC               | Decimal-<br>to-binary              | Data de-<br>fining          | Normal   | DEC                           | Decimal subfields   | Decimal characters con-<br>verted into binary  |
| END               | End of as-<br>sembly<br>pass       | Assembly<br>control         | Not Applicable   | END                           | Address for trans-<br>fer of control,<br>following loading<br>process | Terminates assembly pass   |

APPENDIX A (Cont)  
SUMMARY OF DAP PSEUDO-OPERATIONS

| Oper.<br>Mnemonic | Meaning                   | Class                  | Contents of Fields           |           |   | Effect  |
|-------------------|---------------------------|------------------------|------------------------------|-----------|---|---|
|                   |                           |                        | Location                     | Operation | Variable  |   |
| EQU               | Equals                    | Symbol defining        | Normal (See "Effect" column) | EQU       | Previously defined absolute or relocatable expression | Causes DAP to assign the value and mode of the expression in the variable field to the symbol in the location field |
| EXD               | Enter extend mode         | Loader control         | Not Applicable               | EXD       | Ignored   | Subsequent instructions assembled in extended addressing mode   |
| FIN               | Finish                    | Assembly control       | Not Applicable               | FIN       | Ignored   | Punch out literals  |
| LOAD              | Load mode                 | Assembly control       | Not Applicable               | LOAD      | Ignored   | Subsequent instructions assembled in load mode  |
| LIST              | Generate listing          | List control           | Not Applicable               | LIST      | Ignored   | Causes print-out of source and object programs, side-by-side  |
| LXD               | Leave extend mode         | Loader control         | Not Applicable               | LXD       | Ignored   | Subsequent instructions assembled in normal addressing mode   |
| MOR               | More                      | Assembly control       | Not Applicable               | MOR       | Address for transfer of control                       | Interrogate sense switches to determine type of assembly control  |
| NLST              | No listing                | List control           | Not Applicable               | NLST      | Ignored   | Inhibits program print-out  |
| OCT               | Octal-to-binary           | Data defining          | Normal                       | OCT       | Octal subfields                                       | Octal characters converted into binary  |
| ORG               | Origin                    | Assembly control       | Normal                       | ORG       | Previously defined absolute or relocatable expression | Value and mode of expression in variable field is equivalent and location counter is set accordingly                |
| PZE               | Plus ZERO                 | Special mnemonics code | Normal                       | PZE       | Normal  | ZEROs put into op-code  |
| REL               | Relocatable mode          | Assembly control       | Not Applicable               | REL       | Ignored   | Subsequent instructions assembled in relocatable mode   |
| SETB              | Set base mode             | Loader control         | Normal                       | SETB      | Previously defined absolute or relocatable expression | Specify a sector other than ZERO as the base sector   |
| SUBR              | Entry point               | Program linking        | Ignored                      | SUBR      | Name of subroutine, entry address                     | Punches subroutine name for identification in subroutine library  |
| XAC               | External address constant | Program linking        | Normal                       | XAC       | Name of subroutine                                    | Cause DAP-16 to assemble 16-bit address word defining location outside the program                                  |

APPENDIX B  
DAP OPERATION CODES  
(Listed in Alphabetical Order)







| <u>Mnemonic</u> | <u>Octal Code</u> | <u>Instruction</u>                       | <u>Type</u> | <u>Configuration(s)</u>          |
|-----------------|-------------------|--|-------------|----------------------------------|
| ACA             | 141216            | Add C to A                               | G           | DDP-116 and DDP-516              |
| ADD             | 06                | Add                                      | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| ALR             | 0416              | Logical Left Rotate                      | SH          | DDP-116, DDP-416,<br>and DDP-516 |
| ALS             | 0415              | Arithmetic Left Shift                    | SH          | DDP-116, DDP-416,<br>and DDP-516 |
| ANA             | 03                | AND to A                                 | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| AOA             | 141206            | Add One to A                             | G           | DDP-116 and DDP-516              |
| ARR             | 0406              | Logical Right Rotate                     | SH          | DDP-116, DDP-416,<br>and DDP-516 |
| ARS             | 0405              | Arithmetic Right Shift                   | SH          | DDP-116, DDP-416,<br>and DDP-516 |
| CAL             | 141050            | Clear A, Left Half                       | G           | DDP-516                          |
| CAR             | 141044            | Clear A, Right Half                      | G           | DDP-516                          |
| CAS             | 11                | Compare                                  | MR          | DDP-116 and DDP-516              |
| CHS             | 140024            | Complement A Sign                        | G           | DDP-116 and DDP-516              |
| CMA             | 140401            | Complement A                             | G           | DDP-116 and DDP-516              |
| CRA             | 1400040           | Clear A                                  | G           | DDP-116, DDP-416,<br>and DDP-516 |
| CSA             | 140320            | Copy Sign and Set Sign Plus              | G           | DDP-116 and DDP-516              |
| ENB             | 000401            | Enable Program Interrupt                 | G           | DDP-116, DDP-416,<br>and DDP-516 |
| ERA             | 05                | Exclusive OR to A                        | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| HLT             | 000000            | Halt                                     | G           | DDP-116, DDP-416,<br>and DDP-516 |
| IAB             | 000201            | Interchange A and B                      | G           | DDP-116 and DDP-516              |
| ICA             | 141340            | Interchange Characters in A              | G           | DDP-516                          |
| ICL             | 141140            | Interchange and Clear Left Half<br>of A  | G           | DDP-516                          |
| ICR             | 141240            | Interchange and Clear Right<br>Half of A | G           | DDP-516                          |
| IMA             | 13                | Interchange Memory and A                 | MR          | DDP-116 and DDP-516              |
| INA             | 54                | Input to A                               | IO          | DDP-116, DDP-416,<br>and DDP-516 |


| <u>Mnemonic</u> | <u>Octal Code</u> | <u>Instruction</u>                 | <u>Type</u> | <u>Configuration(s)</u>          |
|-----------------|-------------------|------------------------------------|-------------|----------------------------------|
| INH             | 001001            | Inhibit Program Interrupt          | G           | DDP-116, DDP-416,<br>and DDP-516 |
| INK             | 000043            | Input Keys                         | G           | DDP-516                          |
| IRS             | 12                | Increment, Replace and Skip        | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| JMP             | 01                | Unconditional Jump                 | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| JST             | 10                | Jump and Store Location            | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| LDA             | 02                | Load A                             | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| LDX             | 15                | Load X                             | MR          | DDP-516                          |
| LGL             | 0414              | Logical Left Shift                 | SH          | DDP-116, DDP-416,<br>and DDP-516 |
| LGR             | 0404              | Logical Right Shift                | SH          | DDP-116, DDP-416,<br>and DDP-516 |
| LLL             | 0410              | Long Left Logical Shift            | SH          | DDP-116 and DDP-516              |
| LLR             | 0412              | Long Left Rotate                   | SH          | DDP-116 and DDP-516              |
| LLS             | 0411              | Long Arithmetic Left Shift         | SH          | DDP-116 and DDP-516              |
| LRL             | 0400              | Long Right Logical Shift           | SH          | DDP-116 and DDP-516              |
| LRR             | 0402              | Long Right Rotate                  | SH          | DDP-116 and DDP-516              |
| LRS             | 0401              | Long Arithmetic Right Shift        | SH          | DDP-116 and DDP-516              |
| NOP             | 101000            | No Operation                       | G           | DDP-116, DDP-416,<br>and DDP-516 |
| OCP             | 14                | Output Control Pulse               | IO          | DDP-116, DDP-416,<br>and DDP-516 |
| OTA             | 74                | Output From A                      | IO          | DDP-116, DDP-416,<br>and DDP-516 |
| OTK             | 171020            | Output Keys                        | IO          | DDP-516                          |
| RCB             | 140200            | Reset C Bit                        | G           | DDP-116 and DDP-516              |
| SCB             | 140600            | Set C Bit                          | G           | DDP-116 and DDP-516              |
| SKP             | 100000            | Unconditional Skip                 | G           | DDP-116, DDP-416,<br>and DDP-516 |
| SKS             | 34                | Skip if Ready Line Set             | IO          | DDP-116, DDP-416,<br>and DDP-516 |
| SLN             | 101100            | Skip if (A <sub>16</sub> ) is ONE  | G           | DDP-116 and DDP-516              |
| SLZ             | 100100            | Skip if (A <sub>16</sub> ) is ZERO | G           | DDP-116 and DDP-516              |
| SMI             | 101400            | Skip if A Minus                    | G           | DDP-116, DDP-416,<br>and DDP-516 |
| SMK             | 74                | Set Mask                           | IO          | DDP-116, DDP-416,<br>and DDP-516 |
| SNZ             | 101040            | Skip if A Not ZERO                 | G           | DDP-116, DDP-416,<br>and DDP-516 |
| SPL             | 100400            | Skip if A Plus                     | G           | DDP-116, DDP-416,<br>and DDP-516 |

| <u>Mnemonic</u> | <u>Octal<br/>Code</u> | <u>Instruction</u>              | <u>Type</u> | <u>Configuration</u>             |
|-----------------|-----------------------|---------------------------------|-------------|----------------------------------|
| SRC             | 100001                | Skip if C Reset                 | G           | DDP-116 and DDP-516              |
| SR1             | 100020                | Skip if Sense Switch 1 is Reset | G           | DDP-116 and DDP-516              |
| SR2             | 100010                | Skip if Sense Switch 2 is Reset | G           | DDP-116 and DDP-516              |
| SR3             | 100004                | Skip if Sense Switch 3 is Reset | G           | DDP-116 and DDP-516              |
| SR4             | 100002                | Skip if Sense Switch 4 is Reset | G           | DDP-116 and DDP-516              |
| SSC             | 101001                | Skip if C Set                   | G           | DDP-116 and DDP-516              |
| SSM             | 140500                | Set Sign Minus                  | G           | DDP-116 and DDP-516              |
| SSP             | 140100                | Set Sign Plus                   | G           | DDP-116 and DDP-516              |
| SSR             | 100036                | Skip is no Sense Switch Set     | G           | DDP-116 and DDP-516              |
| SSS             | 101036                | Skip if any Sense Switch is Set | G           | DDP-116 and DDP-516              |
| SS1             | 101020                | Skip if Sense Switch 1 is Set   | G           | DDP-116 and DDP-516              |
| SS2             | 101010                | Skip if Sense Switch 2 is Set   | G           | DDP-116 and DDP-516              |
| SS3             | 101004                | Skip is Sense Switch 3 is Set   | G           | DDP-116 and DDP-516              |
| SS4             | 101002                | Skip if Sense Switch 4 is Set   | G           | DDP-116 and DDP-516              |
| STA             | 04                    | Store A                         | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| STX             | 15                    | Store X                         | MR          | DDP-516                          |
| SUB             | 07                    | Subtract                        | MR          | DDP-116, DDP-416,<br>and DDP-516 |
| SZE             | 100040                | Skip if A ZERO                  | G           | DDP-116, DDP-416,<br>and DDP-516 |
| TCA             | 140407                | Two's Complement A              | G           | DDP-116 and DDP-516              |



APPENDIX C  
DDP-116 and DDP-516 OPTION OPERATION CODES  
(Listed in Alphabetical Order)

| <u>Mnemonic</u>   | <u>Octal Code</u> | <u>Instruction</u>             | <u>Type</u> | <u>Option</u>              |
|---|-------------------|--------------------------------|-------------|----------------------------|
| DAD    | 06                | Double Precision Add           | MR          | High-Speed Arithmetic Unit |
| DBL    | 000007            | Enter Double Precision Mode    | G           | High-Speed Arithmetic Unit |
| DIV   | 17                | Divide                         | MR          | High-Speed Arithmetic Unit |
| DLD    | 02                | Double Precision Load          | MR          | High-Speed Arithmetic Unit |
| DSB    | 07                | Double Precision Subtract      | MR          | High-Speed Arithmetic Unit |
| DST    | 04                | Double Precision Store         | MR          | High-Speed Arithmetic Unit |
| DXA   | 000011            | Disable Extended Mode          | G           | Extended Addressing        |
| ERM   | 001401            | Enter Restricted Mode          | G           | Memory Lockout             |
| EXA   | 000013            | Enable Extended Mode           | G           | Extended Addressing        |
| MPY   | 16                | Multiply                       | MR          | High-Speed Arithmetic Unit |
| NRM   | 000101            | Normalize                      | G           | High-Speed Arithmetic Unit |
| RMP   | 000021            | Reset Memory Parity Error      | G           | Memory Parity              |
| SCA   | 000041            | Shift Count to A               | G           | High-Speed Arithmetic Unit |
| SGL  | 000005            | Enter Single Precision Mode    | G           | High-Speed Arithmetic Unit |
| SPN   | 100200            | Skip if No Memory Parity Error | G           | Memory Parity              |
| SPS   | 101200            | Skip if Memory Parity Error    | G           | Memory Parity              |

 Applicable to DDP-516 only.



# Honeywell

## COMPUTER CONTROL DIVISION

### FRANCE

Honeywell SA,  
Computer Control Division,  
12 rue Avaulée,  
92 Malakoff.  
☎ 253.9620, 735.8300  
Telex : 26.013

### GERMANY

Honeywell GmbH,  
Computer Control Division,  
6050 Offenbach/Main,  
Kaiserleistrasse 55.  
☎ 8.0641  
Telex : 41-52758

### GREAT BRITAIN

Honeywell Ltd.,  
Computer Control Division,  
53 Clarendon Road,  
Watford, Herts.  
☎ Watford 42391  
Telex : 934227

### NETHERLANDS

Honeywell NV,  
Computer Control Division,  
Egelenburg 150-152,  
Amsterdam-Buitenveldert.  
☎ 020-429666  
Telex : 13066

### SWITZERLAND

Honeywell AG,  
Computer Control Division,  
8008 Zürich,  
Dufourstrasse 47.  
☎ (051) 47.44.00  
Telex : 53.561