LEAP Assembler Manual

Lockheed Electronics

MAC 16

LEAP Assembler Manual

Lockheed Electronics

MAC 16

TM3013501103
July 1969 Second Edition

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

Section 1 Introduction

This manual contains a discussion of the Lockheed Electronics Assembly Program (LEAP) for the MAC 16 computer. Included are discussions of LEAP and its purpose, the LEAP language and the rules for using the language, programming examples using the LEAP language, and instructions for preparing a program for assembly. Assumption is made that the reader is familiar with the MAC 16 (described in the MAC 16 Programmer's Manual) and is generally familiar with computer programming at the assembly language level.

GENERAL

An assembly program is a language translator that translates one language (called the source language) into another language (called the object language). The process by which the translation takes place is called the assembly process. The source language is in symbolic form, a form easily understood (and used) by the programmer. The object language is in numeric form, a form understood by the computer.

As in all languages, the LEAP source language has grammatical rules that must be followed so that the "sender" (the programmer) can be understood by the "receiver" (the LEAP program). The entire set of grammatical rules of language is called the syntax of the language. In later sections, the LEAP syntax is covered in detail; this section is concerned primarily with general descriptive information and definitions of terms.

Programmer Options

The LEAP assembly language provides several basic programmer options that a user may exercise. First the LEAP language contains two assembly modes, and second, two formats for object code generation. The two modes of assembly are (1) absolute and (2) relocatable. Through the use of pseudo-operations the programmer can change a mode at will. For example, to access the MAC 16 executive page in memory, an absolute memory location is specified; whereas, unless the program is assembled in bootstrap format, the majority of memory reference instructions refer to locations that are of relocatable mode.

The two object code formats are (1) bootstrap and (2) extended. A program object code format cannot be changed during assembly. Basically, what makes assembling in bootstrap format different from assembling in extended format is that in bootstrap format the programmer must do his own memory reference depaging, and also, he cannot use external references or use the relocation mode of assembly. See Section 3 for an explanation. Under extended format, depaging is automatic during the loading process, and also, both external and relocatable references are allowed.

Features

Some LEAP features follow:

1. Two-pass assembler
2. The programmer completely controls depaging. If the programmer wishes, the assembler/loader automatically depages programs of any size
3. As many as 200 symbols and/or literals can be defined within the symbol table of the assembler operating on a basic 4K MAC 16
4. Pseudo-operations are provided for:
   a. defining constants of various types
      1. Single and double precision, fixed-point (with scaling)
      2. Single and double precision, floating-point
      3. Hexadecimal
      4. Text

b. conditional assembling
c. defining COMMON data storage
d. reserving and/or clearing areas of computer memory
e. defining external programs and data
f. controlling the format of the object code (extended or bootstrap)
g. controlling the program listing

5. Expressions are allowed, including parenthetical sub-expressions, with a full range of operators
6. Literals are allowed
7. 16 error flags (up to 4 printed with each source line)
8. Macros
9. Symbol-table dump (in alphabetical order).

THE ASSEMBLY PROCESS

LEAP reads the source language (one line or statement at a time) and produces the object language. The Loader reads the object language, does whatever processing is necessary (depaging, relocating, etc.), and stores the object program in memory.

To assemble a program, LEAP reads the source program two times. Each reading is called a pass; thus, LEAP is a two-pass assembler. Object code is produced during pass 2.

Pass 1

The purpose of pass 1 is to:

1. Develop a table of defined symbols and assign memory addresses (values) to them
2. Develop a table of external references
3. Develop a table of literals and assign values to them
4. Define Macros.

At the start of pass 1, the two location counters used by LEAP are set to zero (Execution Location Counter, and Common Counter); a source line is then read. If the source line is a comment, the source line is passed over and LEAP reads the next source line. If the OPERATION field contains a symbol-defining pseudo-operation, the symbol appearing in the LOCATION field is entered into the symbol table with the value of the VARIABLE field and LEAP reads the next source line. Each time a pseudo-operation that modifies one of the Location Counters is encountered, the proper Location Counter is modified according to what appears in the VARIABLE field. If a symbol appears in the LOCATION field of one of these pseudo-operations, a value may be assigned before or after the modification, depending on the pseudo-operation. All MAC 16 instructions cause the LEAP Location Counter to be increased by one. If a symbol is encountered in the LOCATION field of a MAC 16 instruction, it is assigned the value of the Location Counter before the increase. If a macro definition is encountered, the macro is stored for a macro call. When a call is encountered for the macro, it is inserted in the program in place of its call. This process continues until the end of the program (END pseudo-operation) is encountered. Pass 2 now starts.

Pass 2

LEAP starts reading the source program again. If the operation is a pseudo-operation controlling the assembly process, the proper function is performed. If the operation is a MAC 16 machine instruction, the proper operation code is inserted into the object code. The first subfield is evaluated in the VARIABLE field and its value is inserted into the object code (if it is legal to have a value in the VARIABLE field). The second subfield is evaluated and if not zero, the index bit is set to one in the object code (if it is legal to have indexing for this instruction). If the first subfield of the VARIABLE field was preceded by an asterisk, the indirect address bit is set to one in the object code (if indirect addressing is legal for this instruction). The source line and the object code is now transferred to the program listing device. The object code is placed in a table and when the table is full, the code is transferred to the object producing device.

This process continues until the end of the source program (END pseudo-operation) is once again encountered. The assembler then produces the object code for literals and the process is complete.

Section 2

LEAP Language Syntax

This section contains the grammatical rules (syntax) of the LEAP language. Rules for forming symbols, expressions, data, operations, etc. are discussed along with some possible error conditions and resultant consequences. This section, along with Sections 3 and 4 provides a complete description of the entire LEAP language. This Section is primarily concerned with statement format; Sections 3 and 4 contain descriptions of the pseudo-operations and machine operations recognized by LEAP.

GENERAL

Throughout this manual, little reference is made to input/output media. LEAP is not concerned with external representation of the source input, or the object and listing output but communicates with the outside world via an elementary executive. (LEAP asks for a source line and is not concerned with where the executive received the source line; in the same manner, the object code and listing code are given to the executive for processing.)
The internal format for the source, object, and listing are:

SOURCE—80—column card image
OBJECT—72—column binary card image
LISTING—108—column print image, plus
carriage control characters

An example of the coding sheet is shown on page 2-3.

Statements are composed of fields; each containing a constant, a symbol, an expression, or two or more of these separated by commas. The assembler evaluates constants and expressions, and saves a table of the symbols and the values assigned to these symbols when the programmer (or the assembly program itself) defines them.

The four fields composing a statement are: the LOCATION field, the OPERATION field, the VARIABLE field, and the COMMENT field, in that order. Fields are separated by one or more blanks. The LOCATION field begins with the first character of the statement, and is terminated by the first blank. The OPERATION field immediately follows the LOCATION field and is terminated with a blank. The VARIABLE (operand) field is defined by the first non-blank character following the OPERATION field if this character is within 5 blank characters of the OPERATION field, and is terminated by the first blank character or by an end of record signal. (The end of record signal on paper tape is defined as a carriage return-line feed.) If the first non-blank character after the OPERATION field is greater than 5 blank characters away, or if there is a non-blank character following the VARIABLE field, the COMMENTS field is defined. The COMMENTS field is terminated either by the end-of-record signal or by the eightieth character. An entire line of comments can be entered by writing an asterisk in the first character position of the line. Fields are usually written in fixed positions on a coding sheet; however, any format that satisfies the above definition is permissible. Free form tolerance is of greatest advantage in overcoming a common keypunching error; that is, starting a field in the wrong column or character position of the line.

THE CHARACTER SET

The character set used by LEAP (and the MAC 16) is the USA Standard Code for Information Interchange (USASCII) set. There is syntactic significance to some of the characters, but this is discussed later (see ELEMENTS later in this section). The USASCII character set is listed in APPENDIX A.

FIELDS

The LEAP CODING FORM is used to assist the programmer in writing a program in a consistent format acceptable to LEAP. Each line of the form is used for

writing one symbolic line of the program (called a source statement). The source statement is divided into five fields:

LOCATION
OPERATION
VARIABLE
COMMENTS
IDENTIFICATION

No provisions in LEAP are made for continuation of source statements except as outlined in macro definition (Section 3).

The LOCATION Field

The LOCATION field normally is used for a symbol by which a source statement can be referenced from other parts of the source program (often referred to as the symbolic location). The LOCATION field has a special meaning for some pseudo-operations and is ignored by others; the discussion with each pseudo-operation (Section 3) specifically states the use of the LOCATION field.

A legal symbol within the LOCATION field consists of one or more (up to a maximum of six) alphanumeric characters (A through Z, 0 thru 9, and colon (:)), the first of which must be alphabetic. The symbol must start in the first character position of the LOCATION field and is terminated by the first blank.

The special character colon (:) is considered a numeric symbol by the syntax of the LEAP assembler when used within the LOCATION or VARIABLE fields.

Symbols

Symbols consist of alphabetic characters, (letters A through Z), numeric characters (numbers 0 through 9 and colon (:)) or a combination of alphabetic and numeric characters. A symbol may be formed of one to six characters, but the first character of every symbol must be alphabetic.

The special character colon (:) is considered a numeric character by the syntax of LEAP.

Example:

LEGAL
ALPHA
PI
A239B
P68475
X
ZAP:U
ILLEGAL (Error Code "S")
3P14 Does not begin with an alphabetic character
AB.CD Illegal character used (.)
ALPHABET More than six characters
'1FF Illegal character used (')
:AB doesn't begin with an alphabetic character

If an asterisk (*) appears in card column 1 (and only in card column 1) of the LOCATION field, the entire source line is treated as a comment. That is, it will be printed on the listing, but otherwise ignored by the assembler. This allows the programmer to freely document his program listing without adding to the size of his object code.

The following are the only entries allowed in the LOCATION field:

all blank
legal symbol
*in column 1

Violation of this rule causes the error code "S" to be printed on the listing.

The remaining discussion assumes that no asterisk is written in card column 1 (the remainder of the card is ignored, if a comment).

The OPERATION Field

The OPERATION field is used for entering the mnemonic for a desired operation. This operation can be the mnemonic for a pseudo-operation (see Section 3) or a machine operation (see Section 4). When the mnemonic represents one of the MAC 16 machine operations, LEAP inserts the appropriate code for that operation into the object code produced by the source statement. When the mnemonic represents one of the LEAP pseudo-operations, the instructed action is taken. Appendices B and C summarize the mnemonics that are accepted by LEAP in the OPERATION field. The contents of the OPERATION field are optional i.e., a macro (see Section 3). The OPERATION field starts with the first non-blank character following the LOCATION field and is terminated by the first blank character encountered.

A mnemonic entry within the OPERATION field consists of one to six alphanumeric characters (A thru Z, 0 thru 9, and colon (:)), the first of which must be alphabetic.

The remaining characters must conform to the syntax for either a mnemonic or a valid pseudo-operation, a machine operation or a predefined macro. Otherwise, the statement is flagged with an error code "0".

LEAP CODING FORM

PROGRAMMER | DATE | PAGE OF

PROGRAM | CHARGE

	LOCATION	OPERATION	VARIABLE	COMMENTS	IDENTIFICATION
1	* USE	ØF THE	VARIABLE FIELD (MEMØRY REFERENCE ØNLY)		
2	* THE	FØLLØWING	ILLUSTRATES AN INDEXED INSTRUCTIØN		
3		LDA	SYMBL,1		
4	*				
5	* THE	FØLLØWING	ILLUSTRATES AN INDIRECT INSTRUCTIØN		
6		LDA	*SYMBØL		
7	*				
8	* THE	FØLLØWING	ILLUSTRATES AN INDIRECT-INDEXED		
9			INSTRUCTIØN		
10		LDA	*SYMBL,1		
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					

The VARIABLE Field

The VARIABLE field or operand is so called because of its function as modifier to the operation. The VARIABLE field starts with the first non-blank character following the OPERATION field and is terminated with the first blank encountered (or column 80, whichever is first). It is most commonly used for specifying the variable part of a machine instruction (address, M, N, amount of shift, etc.), and the index register flag. When used for these purposes, the VARIABLE field is actually broken down into subfields.

For memory-reference instructions, there are two subfields: the address subfield and the X (index) subfield. The programmer indicates the separation of these two subfields by placing a comma (,) between them.

Section 3 includes a description for the use of the VARIABLE field on various pseudo-operations; Section 4 describes the uses for the various machine operations.

If an asterisk (*) preceeds the mnemonic in the VARIABLE field, it is interpreted to mean that the instruction indirect address tag, I, is to be set to a one. This is allowed with all memory reference operations. If the operation is not a memory reference type, the error code "I" is printed on the listing.

LEAP terminates the scan of the VARIABLE field when the first blank is encountered (exceptions: USASCII data and the TXT and TITLE pseudo-operations).

The COMMENTS Field

The COMMENTS field is used for any remarks that the programmer cares to write. The contents of the COMMENTS field are printed on the program listing, but are otherwise ignored by the assembler.

The COMMENTS field starts with the first blank following the VARIABLE FIELD (except as noted above). Although the starting position of the COMMENTS field is variable, the program listing is less confusing if the programmer always starts a comment in the same column. By following this rule, a predictable separation exists between the program instructions and the commentary, thus, making a more readable listing.

The IDENTIFICATION Field

The IDENTIFICATION field may be used for record identification (if cards are used) and is considered an extension of the COMMENTS field. Otherwise, IDENTIFICATION is treated in the same manner as described for COMMENTS.

EXPRESSIONS

It is often desirable to specify a value in the VARIABLE field that is generated by combining more than one value. LEAP provides the programmer with this capability by allowing expressions in the VARIABLE field. An expression is composed of elements and operators; it can be composed of a single element, or n elements separated by n-1 operators. The length of an expression is limited only to the size of the VARIABLE field.

If two or more elements appear in an expression, all of the elements in the expression must be single precision; i.e., a value that is not larger than 16 bits. The only time an element is multiple precision (e.g., double precision numbers, or text string of more than two characters, etc.) is when the element is in the VARIABLE field of a multiple precision DC or TXT pseudo-operation. Illegal use of multiple precision elements causes the error code "C" to be printed on the listing.

Elements

An element is an operand in an expression. An element may be a symbol, a datum, or the asterisk.

As stated, LEAP places special syntactic significance on some characters in the character set. Some of this significance is associated with the first character of an element.

If the first character of an element is:

A thru Z: the element is a symbol

0 thru 9 or the decimal point: the element is a decimal number

$: the element is a hexadecimal number

': the element is USASCII data.

Value Assignment

A symbol normally has an assigned value when the symbol is encountered in the LOCATION field of a source line. When an undefined symbol (not in the LOCATION field of any source line) is encountered in the VARIABLE field, LEAP assigns a value of zero and causes the error code "U" to be printed with the line in which the symbol was encountered.

If a symbol does not have a unique value (i.e., it was encountered in the LOCATION field of more than one source line), it is considered to be a multiple definition. The value of the symbol is the value assigned by LEAP the last time the symbol was encountered in the LOCATION field; subsequent encounters change the value. Multiple definitions cause the error code "D" to be printed with the line in which the symbol is encountered in the LOCATION field. If a source statement refers to the multiply defined symbol in the VARIABLE field, the error code "M" is printed with the line in which the symbol is encountered.

The value of a symbol encountered in a pseudo-operation LOCATION field is the value and type of the LEAP LOCATION counter at the time the source line is encountered with two exceptions. These exceptions are the EQU and EQUR pseudo-operations (see Section 3 for further explanation.)

Mode Assignment

The mode of a symbol is dependent on the mode of the value assigned to the symbol. A value may be either absolute, external, or relocatable.

Absolute

A symbol is absolute if the value assigned to the symbol is absolute. This can be done in two ways:

1. LEAP assigns the value of the LOCATION counter and the mode of assembly is absolute; or,
2. the value is assigned through the use of a symbol-defining pseudo-operation and the expression in the VARIABLE field is absolute.

External

A symbol is external if it appears in the VARIABLE field of an EXTRN pseudo- operation.

Relocatable

A symbol is relocatable if the value assigned to the symbol is relocatable.

This can be done in three ways:

1. LEAP assigns the value of the LOCATION counter and the mode of assembly is relocatable; or,
2. the value is assigned through the use of a symbol-defining pseudo-operation and the expression in the VARIABLE field is relocatable; or
3. The symbol appears in the LOCATION field of a COMN pseudo-operation.

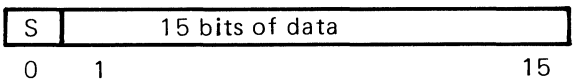
Data

LEAP will process three different data notations as elements: decimal, hexadecimal, and USASCII text.

DECIMAL

Single Precision, Fixed-Point (4.5 decimal digits)

SINGLE PRECISION, FIXED-POINT (1 word)



A single precision, fixed-point decimal number requires one computer word (sign and 15 bits) and is written in two parts: the numeric part and the scaling part.

The numeric part of the single precision, fixed-point number is a signed or unsigned decimal number, with or without a decimal point. If no sign is specified, the number is assumed to be positive. If no decimal point is specified, it is assumed to be immediately to the right of the final digit of the numeric part (a decimal integer).

The scaling part of the single precision, fixed-point number is composed of the binary scale factor and the decimal scale factor. The binary scale factor may be omitted if no decimal point is included and must be included when the binary scale factor is the letter B followed by a signed or unsigned decimal integer. The binary scale factor specifies the position of the understood binary point relative to the machine binary point (between bit positions 0 and] of the computer word). If the binary scale factor is positive, the understood binary point is to the right of the machine binary point, if negative, it is to the left of the machine binary point.

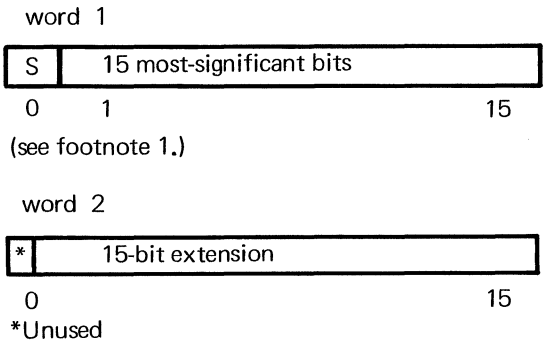
The decimal scale factor may be omitted, but if included, it is the letter E followed by a signed or unsigned decimal integer. The decimal scale factor specifies the location of the decimal point in the numeric part relative to the written decimal point (if no decimal point had been written, it is assumed to be immediately to the right of the final digit). If the decimal scale factor is positive, the decimal point is to the right; if negative, the decimal point is to the left.

The binary scale factor may appear before the decimal scale factor, or vice versa, but the scale factors must appear after the numeric part (error code "C", if not).

If information is lost in the most significant part of the word during conversion (improper scaling), the error code "C" is printed on the listing.

Double Precision, Fixed-Point (9.4 decimal digits)

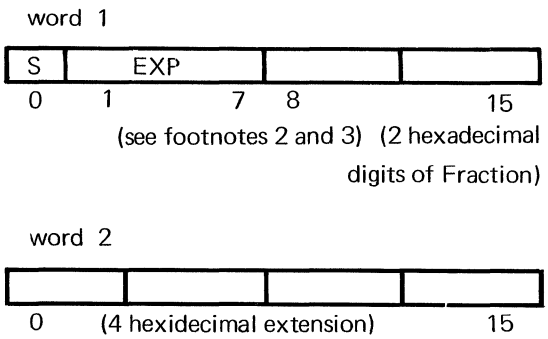
DOUBLE PRECISION, FIXED-POINT (2 words)



A double precision, fixed-point decimal number requires two computer words (sign and 31 bits). A double precision, fixed-point number is written the same way as the single precision number, except that two B's are written for the binary scale factor or two E's are written for the decimal scale factor (or both).

Single Precision, Floating-Point (7 significant decimal digits)

SINGLE PRECISION, FLOATING-POINT (2 words)

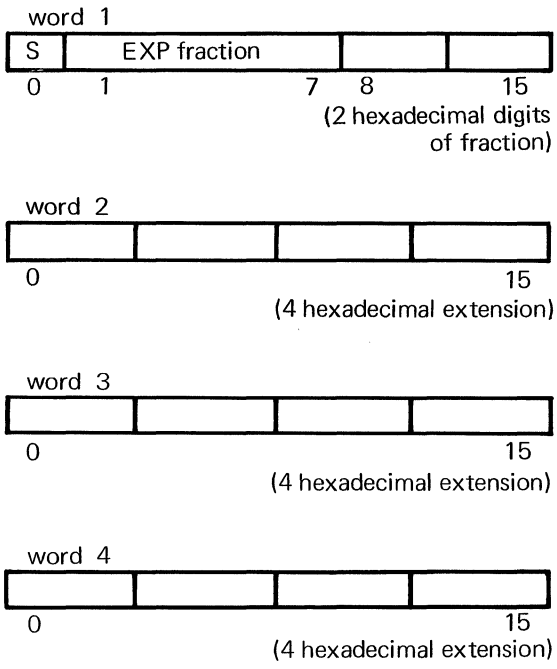


A single precision, floating-point decimal number requires two computer words (sign, 7-bit exponent, and 6 hexadecimal digits of data) and is written in two parts: the numeric part and the scaling part.

The numeric part of the single precision, floating-point number is a signed or unsigned decimal number, with a decimal point. If no sign is specified, the number is assumed to be positive. If no decimal point is specified, the number is assumed to be an integer.

The scaling part of the single precision, floating-point number is the decimal scale factor. The decimal scale factor specifies the location of the decimal point relative to the written decimal point in the numeric part. If the decimal scale factor is positive, the decimal point is to the right; if negative, the decimal point is to the left.

Double Precision, Floating-Point (16.7 significant decimal digits)



A double precision, floating-point decimal number requires four computer words (sign, 7-bit hexadecimal exponent, 14 hexadecimal digits of data).

A double precision, floating-point number is written the same way as the single precision number, except that a decimal scale factor must be specified, and two E's must be written for the decimal scale factor.

Notes:

1. S is the sign
2. EXP is the 7-bit exponent (power of 16), offset by 40_{16}. The decimal range of the hexadecimal exponent is +63 to -64 (approximately 10_{+19} to 10_{-19}).
3. Fraction is the normalized hexadecimal fraction; extensions are 4 hexadecimal digits.

LEAP CODING FORM

PROGRAMMER | DATE | PAGE OF

PROGRAM | CHARGE

	LOCATION	OPERATION	VARIABLE	COMMENTS	IDENTIFICATION
1	* THE FØLLØWING ILLUSTRATES THE DEFINITIØN ØF				
2	* DECIMAL DATA				
3	* EXAMPLE 1		SINGLE PRECISIØN,FIXED-PØINT		
4	TAG1	DC	1.B15	EQUALS HEX 0001	
5	*				
6	* EXAMPLE 2		DØUBLE PRECISIØN,FIXED-PØINT		
7	TAG2	DC	.1EE1B31	EQUALS HEX 0000 0001	
8	*				
9	* EXAMPLE 3		SINGLE PRECISIØN,FLØATING-PØINT		
10	TAG3	DC	-1	EQUALS HEX C110 0000	
11	*				
12	* EXAMPLE 4		DØUBLE PRECISIØN,FLØATING-PØINT		
13	TAG4	DC	1.EE-1	EQUALS HEX 4019 9999 9999 9999	
14	*				
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					

Examples of Decimal Data

NUMBER	FORMAT	AS IT APPEARS ON THE LISTING (HEXADECIMAL)
1	single precision, fixed-point	0001
1.	single precision, floating-point	4110
		0000
1.B15	single precision, fixed-point	0001
.1EE1B31	double precision, fixed-point	0000
		0001
-1.	single precision, floating-point	C110
		0000
1.EE-1	double precision, floating-point	4019
		9999
		9999
		9999

Hexadecimal

A hexadecimal number is represented by the dollar sign ($) followed by a signed or unsigned string of hexadecimal digits. A single precision hexadecimal number has less than five digits; a double precision hexadecimal number has more than four digits, but fewer than nine digits. If a negative sign is specified, the resulting number is two's complement.

USASCII

USASCII text is defined to be the single quote (') followed by a string of characters. Two characters are stored per word. If more than two characters are written, the string will use more than one word. A single quote (') terminates the strings.

Control characters (carriage return, line feed, tab, etc.) and the single quote (') are not allowed in a USASCII text string. If it is necessary to specify any of these a separate DC statement must be used, with the character(s) specified in hexadecimal.

Example of USASCII characters in a DC statement:
DC, 4 'WEDNESDA'

The Asterisk As An Element

A single asterisk (*) may be used as an element in an expression. The value of the asterisk is the value of the LOCATION counter at the time the asterisk is encountered during the expression evaluation. The value mode of the asterisk depends on the current mode of assembly (absolute or relocatable).

Expression Operators

Items are combined using the operators defined in the table below. The table also is a list of hierarchy numbers for determining how the value of an expression is computed. Operations with greater hierarchical value are performed before operations having lesser hierarchical value. Operations with the same hierarchical value are performed from left to right. (Parentheses may be used to change this hierarchical structure.)

Table 3

Hierarchy	Operator	Description
6	%	Logical binary left shift
5	*	Arithmetic product
5	/	Arithmetic quotient
4	+	Arithmetic sum
4	-	Arithmetic difference
3	<	Boolean less than
3	=	Boolean equals
3	>	Boolean greater than
2	&	Logical product (AND)
1	@	Logical sum (OR)
1	!	Logical difference (exclusive OR)

An expression can have a leading minus sign; if so, its value is computed as though a zero preceded the minus sign. Thus - 13% (-2) is evaluated as 0-(13%(-2)), which is not the same as (0-13% (-2).

The value of an item will be right-justified in its generated resultant field and unspecified leading bit positions will contain zeros.

Note that the subexpression item provides for parenthetical expressions; i.e., according to the above rules (A+B)/2 is a valid expression.

Elements And Operators In Expressions

If two or more elements appear in an expression, all of the value are treated as 16-bit values. A multiple precision element must exist alone in an expression and can only be used in the VARIABLE field of a DC or TXT pseudo-operation.

Two consecutive elements are not permitted without an operator separating them (3M does not mean 3*M and will cause the error code "E" to be printed on the listing).

Consecutive operators are not permitted without elements except for the minus sign (-). Any of the elements described in this section may be used in expressions (with the above-noted restriction on multiple precision elements). Certain combinations, described below, have no meaning and are not permitted. The table at the end of this section summarizes the legal and illegal combinations.

Absolute Values

Any 16-bit absolute value can be combined with any other 16-bit absolute value with any operator. Absolute values are:

Symbols with absolute values

Data Elements

Relocatable Values

Only one relocatable value can exist in an expression, unless the combination of relocatable values causes an absolute result. For example, a relocatable value subtracted from a relocatable value produces an absolute result. It is the programmer's responsibility to group the elements in an expression so that the value of the entire expression is dependent on only one relocatable element. Violation of this rule causes the error code "R" to be printed on the listing.

External Values

External values cannot be combined with other expressions elements. Violation of this rule causes the error code "R" to be printed on the listing.

Relocatable and external values are not defined until load time; absolute values are known at assembly time. The use of relocatable and external values is assumed to be associated with a reference to a memory location by the Extended Loader. The following are the only memory-reference values recognized by the Extended Loader.

ABSOLUTE VALUE .	Depage
POSTITIVE RELOCATABLE VALUE	Add relocation constant to value and depage
POSITIVE EXTERNAL VALUE	Add value to value of external symbol and depage.

The Asterisk

If the asterisk is used as an element and it has a relocatable value, the restrictions described above apply for relocatable values in expressions. There is no restriction regarding the use of an asterisk as an absolute value.

The asterisk is also an operator and there is one syntactical ambiguity in the use of the asterisk for either purpose. LEAP evaluates asterisks in the VARIABLE field in the following manner. The first element, if an asterisk, forces LEAP to evaluate the second element. If the second item is a legal operator (+,-,*,/, etc., the value of the first item (*) will be the value and type of the LOCATION counter. If the second item is a legal element the first item (*) will be used to define an indirect call to a location. With this in mind the following example will be easy to decipher.

Example:

```
LDA  **2
```

The value of the expression is the value of the LOCATION counter multiplied by two. If the programmer wished the value to be indirect through LOCATION counter +2, a name (label) should be assigned to the desired location and the coding written as

```
LDA  *NAME
```

Uses of Parentheses In Expressions

LEAP gives the programmer the capability to specify subexpressions within expressions. A subexpression is a string of elements and operators bounded by parentheses. A subexpression starts after a left parenthesis and is closed when a matching right parenthesis is encountered. After evaluating a subexpression, the resulting 16-bit value is treated as an element by LEAP.

Parenthetical subexpressions may be nested within subexpressions. LEAP can evaluate up to five levels of nested parenthetical subexpressions. However, for each left parenthesis encountered, there must be a matching right parenthesis within the expression. If there are more left or right parentheses than the other, the error code "Z" is printed on the listing.

Examples:

(((((A+B)*(C-D))/E) & F) ! $FFFF)+1

(A)+(B)-(C*D)

Note, that redundant parentheses are allowed; A+B-C*D would produce exactly the same results as this example.

ALPHA/(BETA-GAMMA-DELTA)

(BITS/2) & 1

Shifts BITS right 1 place and keeps only low-order bit.

-EXTENT

Forms two's complement

Summary of Legal and Illegal Expressions

The following table lists the legal and illegal combinations of elements in expressions. If an illegal combination is encountered, the error code "R" will be printed on the listing. The table is applicable to values and intermediate values. For example, the table shows that r+r is illegal; therefore, r+r-r would also be illegal; however, r+(r-r) would be interpreted as r+a, which is legal.

LEGAL

a=a	r=r	e=e
a+a=a	a+r=r	
a-a=a	r+a=r	
a*a=a	r-a=r	
a/a=a	r-r=a	
a&a=a		
a@a=a		
a!a=a		

a = absolute; e=external; r=relocatable

ILLEGAL (error code "R")

Any other combination of elements is illegal.
No e or r item may be an element of: *,/,&,@,! operators

LITERALS

A literal is the equals sign(=), followed by any legal expression. Multiple precision elements are not permitted following the equals sign. If a multiple precision element is encountered, the error code "E" is printed on the listing.

The value of a literal is the 16-bit address of the memory location in which the literal is stored.

The value and mode of a literal is assigned at the end of the first pass over the source program. For each literal encountered in the literal table, LEAP assigns the value of the LOCATION counter and then increases the LOCATION counter by one. Mode assignment is dependent on the mode of assembly at the time the END pseudo-operation is encountered. If LEAP is assembling in the absolute mode, the literals will be the absolute mode; if in the relocatable mode, the mode will be relocatable.

Examples:

='AB' requires one word (USASCII string, two characters per word).

=$1FF requires one word (single precision, hexadecimal).

When LEAP is assembling in bootstrap format, literals are used only if the program, including all literals, is less than one page in size.

A literal expression is composed of more than one element (or a single element with a 16-bit value), and is always a 16-bit word containing the value of the expression. Thus, absolute, external, or relocatable values are elements in literals.

Examples:

LDA =*+3 means "load the A-register with a word containing the value of *+3." It does not mean "load the A-register with the word that is three locations away."

EXTRN SINE

LDX =SINE means "load the index register with the external value of SINE." At execution time, this value will be the memory address of where SINE had been placed by the loader.

SUMMARY OF ASTERISK USES

The following are the various uses of the asterisk and their meanings:

1. An asterisk in column 1 of the source statement: treat entire source line as a comment.
2. An asterisk preceding a VARIABLE field mnemonic: set indirect address flag.
3. An asterisk as an element in an expression: current value and mode of the LOCATION counter.
4. An asterisk as an operator: form the product of the adjacent elements.

ERROR CODES

The following is a summary of the error codes and their meanings:

A ADDRESS SUBFIELD
A null expression was encountered in the address subfield (memory reference instructions only). There must always be an entry in the address subfield of memory reference instructions, even if it is just 0 (zero).

B BOOTSTRAP
While running the bootstrap format, either a memory reference instruction has attempted to address a location not in the current page or the base page, or an illegal pseudo-operation has been encountered.

C CONVERSION
A data element cannot be correctly converted. This is usually associated with an illegal character or combination or characters in a data element; however, too many characters in a hexadecimal element or improper scaling of a decimal element can also cause this error.

D DOUBLE DEFINITION
A symbol has been defined more than once in the source program.

E EXPRESSION
An illegal expression has been encountered. There are many reasons for this error; see Section 2 for details.

F FIELD OVERFLOW
A value has been formed that is too large for the intended field. Examples of this are: a value of more than one bit in the Index subfield, a value of more than 4 bits in the M and N subfields, etc.

I INDIRECT ADDRESSING
An asterisk has preceded the VARIABLE field of a non-memory reference instruction.

M REFERENCE TO MULTIPLE DEFINITION
A multipally defined symbol has been referenced in the source statement.

O OPERATION CODE
An illegal mnemonic, or undefined mnemonic has been encountered in the OPERATION field.

P PHASE
A relocatable expression is not permitted, or an expression has an element that has not been previously defined.

R RELOCATABLE
An expression has been encountered with an illegal combination of relocatable and/or external elements.

S SYMBOL
An illegal character has been encountered in the LOCATION field; or a symbol-definition has been attempted without starting the symbol with an alphabetic character; or a symbol has been encountered in the VARIABLE field with too many characters (more than six); or a symbol has been omitted in the LOCATION field for a pseudo-operation requiring a symbol.

U UNDEFINED
A reference has been made to a symbol that is not defined anywhere in the program.

V VARIABLE FIELD
An error has been encountered in the VARIABLE field that cannot be classified more explicitly — this could be a string of characters that has no syntactic meaning; or nothing was entered in the VARIABLE field when an entry is required.

X INDEX SUBFIELD
An index subfield was specified, but nothing was entered; or an index subfield has been encountered for an instruction that does not allow indexing.

Z NON-ZERO LEVEL REDUCTION
An expression has been encountered that does not have matching pairs of parentheses; there must be exactly the same number of left and right parentheses.

Section 3

LEAP Pseudo-Operations

This section contains a description of the pseudo-operations provided in the LEAP language. Examples using the pseudo-operations are included when further clarification is required.

Preceding each pseudo-operation description is a figure showing the format of the LOCATION, OPERATION, and VARIABLE fields.

LOCATION	OPERATION	VARIABLE

The above figure illustrates the format; the fields correspond to the fields defined in Section 2.

Except for the TITLE and PRINT psuedo-operations, the COMMENTS field always starts with the termination of the VARIABLE field (the first space encountered that is not part of text data). To conserve space and the reader's time, the COMMENTS field is not shown with each of the pseudo-operations.

If the word "normal" appears in the LOCATION field of the format figure preceding the pseudo-operation discussion, it means that entry of a symbol in the LOCATION field is optional. If a symbol is entered in the LOCATION field, its value is assigned in the normal manner.

If the word "symbol" appears in the LOCATION field of the format figure preceding the pseudo-operation discussion, it means that there must be a symbol entered in the LOCATION field. If no symbol is encountered, the error code "S" is printed on the listing.

ASSEMBLY CONTROLLING PSEUDO–OPERATIONS

The following paragraphs describe those pseudo-operations provided as part of the LEAP language that allow the programmer to control the format of the object code produced. In addition, pseudo-operations are provided for conditional assembling of source lines and setting the value of the LOCATION counter. A single pseudo-operation is provided for defining the end of the source program.

The pseudo-operations described in this section are:

BOOT – BOOTSTRAP FORMAT
DUP – DUPLICATE NEXT SOURCE LINE
END – END OF SOURCE PROGRAM
ORG – SET PROGRAM ORIGIN
SKIPF – SKIP IF FALSE
SKIPT – SKIP IF TRUE

BOOT – BOOTSTRAP FORMAT

LOCATION	OPERATION	VARIABLE
normal	BOOT	ignored

This pseudo-operation specifies to LEAP that the format of the object code produced should be the format recognized by the bootstrap loader and that the mode of assembly should be absolute for the entire assembly.

There are several reasons why a programmer uses this option. Among these are the following:

1. The programmer is in complete control of program and data storage in memory.
2. All base page linkages can be assigned by the programmer.
3. The Bootstrap Loader program can be used to load memory rather than the much larger Extended Loader.
4. This also saves the added step of using the Extended Loader through its genboot option to generate bootstrap object code.

The BOOT pseudo-operation causes the object program to be completely absolute; no memory-reference instructions may make reference to relocatable or external elements, or elements either outside the page in which the instruction is located or the base page. If a violation of these rules occurs, the error code "B" will be printed on the listing with the source line in which the offending element is encountered. When used, the BOOT pseudo-operation must be the first source statement to be read by LEAP or it will be considered an illegal command to LEAP and the error code "B" will be printed on the listing with the source line. See Section 5 for a description of the bootstrap format.

A summary of illegal pseudo-operations when assembling with he BOOT pseudo-operation are as follows:

1. ORG — when the expression is relocatable;
2. EXTRN
3. ENTRY
4. CLEAR
5. COMN
6. DS — when the expression is relocatable;
7. EQUR

Each of these pseudo-operations requires either relocatable items or special type codes. Since bootstrap format requires absolute assembly and has no provision for relocation, each of these will have the "B" error code printed on the listing with the source line in which they appear.

DUP-Duplicate the Next Source Line

LOCATION	OPERATION	VARIABLE
normal	DUP	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that the next source statement should be duplicated the number of times specified by the expression in the VARIABLE field.

The expression in the VARIABLE field must be absolute and if any symbol appears in the expression, the symbol must have been previously defined (no forward reference); violation of this rule causes the error code "P" to be printed on the listing.

If the source statement that follows the DUP pseudo-operation causes object code to be generated, the LOCATION counter will be increased the appropriate amount for each duplication. If the asterisk is used as an element in the expression, the value of the LOCATION counter for each duplication.

The expression must equal some finite value n. The source statement then appears n times, not n+1 times. An example of this pseudo-operation is found in Section 3. When the expression equals zero, the following source line is not assembled.

END — End of Source Program

LOCATION	OPERATION	VARIABLE
normal	END	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that the end of the source program is reached and that the value of the expression in the VARIABLE field defines the location of the first instruction to be executed at run time. THERE MUST BE ONLY ONE END PSEUDO-OPERATION IN A SOURCE PROGRAM.

When LEAP encounters the END pseudo-operation, the following action takes place:

PASS 1

The Location Counter used for value and mode assignment is dependent on the mode of assembly at the time the END pseudo-operation is encountered. If LEAP is processing absolute statements, the mode and value assignment is absolute; if LEAP is processing relocatable statements, the mode and value is relocatable.

1. The literal table is examined. For each literal found, LEAP assigns the value and mode of the LOCATION counter for reference during pass 2. The LOCATION counter is incremented by one for each literal; thus, the value assigned for later reference is actually the location of the literal.
2. LEAP now prints a message signifying the completion of Pass 1 on the system's tele-printer, and waits for operator action before continuing.

PASS 2

1. A listing of the literals and their values (if any exist) is printed.
2. The literals are output in the proper object code format.

3. A listing of all symbols and their values (in alphabetic order) is printed if the LSTSY pseudo-operation has been encountered.

4. The expression in the VARIABLE field is evaluated. If nothing is entered in the VARIABLE field, LEAP outputs the END/JUMP object code and makes the address of the jump equal to absolute zero. If an expression is entered, the resultant value is the address of the jump.

5. An end-of-assembly message is typed on the systems teleprinter.

ORG — Set Program Origin

LOCATION	OPERATION	VARIABLE
normal	ORG	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that the LOCATION counter should be set to the value of the expression in the VARIABLE field. The LOCATION counter may be set absolute or relocatable depending on whether the expression in the VARIABLE field is absolute or relocatable.

When LEAP encounters the ORG pseudo-operation, the expression in the VARIABLE field is evaluated and the LOCATION counter is set to the value and mode of the expression.

When LEAP is assembling bootstrap formatted object code, the expression in the VARIABLE field must not be relocatable. If the expression is relocatable, the error code "B" is printed on the listing with the source statement.

SKIPF — Skip If False

LOCATION	OPERATION	VARIABLE
normal	SKIPF	first expression, second expression,

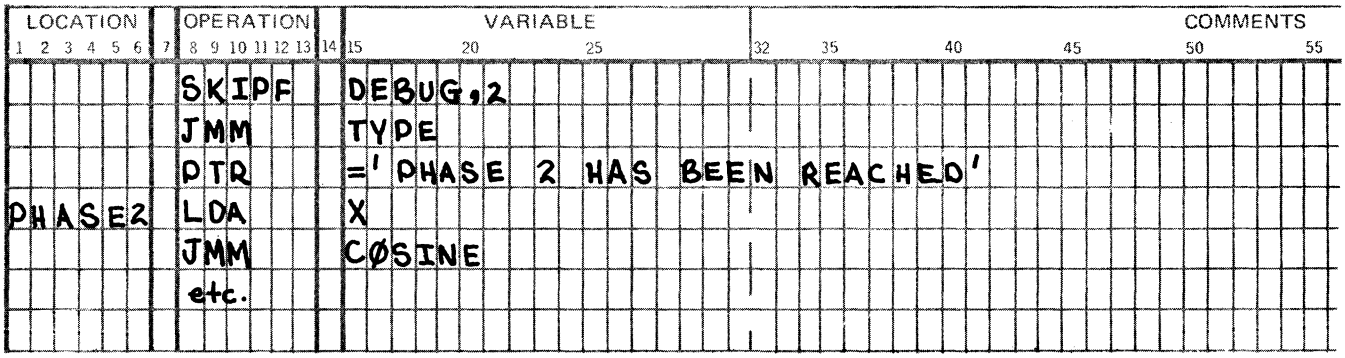
The pseudo-operation gives the programmer the capability to specify to LEAP that a portion of the program should be conditionally excluded from the assembly process, dependent on the first expression being false.

Both of the expressions in the VARIABLE field must be absolute. Any symbols used in the expressions must be previously defined. If either (or both) of these conditions is not met, the error code "P" is printed on the listing and the value of the first expression is set to zero (false).

When LEAP encounters a SKIPF pseudo-operation, it evaluates the first expression in the VARIABLE field. If the value of the expression is not zero (true), the remainder of the line is ignored and LEAP proceeds to the next source line in the normal manner. If the value of the first expression is zero (false), LEAP evaluates the second expression. The resultant value is the number of subsequent source lines skipped. If the LIST pseudo-operation is in effect, the skipped source lines are listed.

In the example below, the programmer specifies that the 2 lines following the SKIPF should not be included in the program, if the value of the symbol, DEBUG, is zero (false). If the value of DEBUG is not zero (true), the 2 lines should be included.

If the programmer chose not to define DEBUG, it would be a phase error (error code "P"), forcing the value of the first expression to be zero, and the next 2 lines would be skipped.



SKIPT — Skip If True

LOCATION	OPERATION	VARIABLE
normal	SKIPF	first expression, second expression

The pseudo-operation gives the programmer the capability to specify to LEAP that a portion of his program should conditionally be excluded from the assembly process, dependent on the first expression being true.

Both of the expressions in the VARIABLE field must be absolute. Any symbols used in the expressions must have been previously defined. If either (or both) of these conditions is not met, the error code "P'. is printed on the listing and the value of the first expression is set to zero (false).

When LEAP encounters a SKIPT pseudo-operation, it evaluates the first expression in the VARIABLE field. If the value of the expression is zero (false), the remainder of the line is ignored and LEAP proceeds to the next source line in the normal manner. If the value of the first expression is not zero (true), LEAP evaluates the second expression. The resultant value is the number of subsequent source lines skipped. If the LIST pseudo-operation is in effect, the skipped source lines are listed.

This coding form example illustrates:

1. use of SKIPT and SKIPF
2. program being modified at assembly time, dependent on the parameter, ARG, and
3. program being modified at execution time, dependent on the same parameter, ARG.

The following discussion explains each line of the example. Assume that ARG is defined earlier in the program and that it has only four values: 0,1,2, or 3.

SKIPT ARG, 2*ARG

If ARG=0, the next, and subsequent lines are assembled in the normal manner. If ARG=1, 2*ARG lines are skipped (2) and assembling resumes at A102. If ARG=2, 4 lines are skipped and assembling resumes at A104. If ARG=3, 6 lines are skipped and assembling resumes at A105+1.

SKIPF ARG-3,9

If ARG=3, none of the lines A100 thru A105 are assembled; therefore, there is no clear loop and all lines down to, but not including, DONE are skipped (if 3-ARG=0, 9 lines are skipped). If ARG=3, normal assembling resumes with the next line.

LOCATION	OPERATION	VARIABLE	COMMENTS
	SKIPT	ARG,2*ARG	
A100	DS	1	
A101	DS	1	
A102	DS	1	
A103	DS	1	
A104	DS	1	
A105	DS	1	
	SKIPF	ARG-3,9	
	LDI	(3-ARG)*2	
	TWA		
	STA	COUNT	
	LDX	=A105	
	CLA		
LOOP	STA	0,1	
	DNX	1	
	INC	COUNT	
	JMP	LOOP	
DONE	etc.		

LDI (3-ARG)*2

The value loaded into the A-register (immediate) is dependent on the value of ARG. If ARG=0, the value loaded is 6,(3-0)*2=6 with 6 the number of locations to be cleared (if ARG=0, none of the lines A100 thru A105 are skipped). If ARG=1, the value loaded is 4; if ARG=2, the value is 2. If ARG=3, this instruction or the next 8 instructions are not included in the program (the SKIPF would have skipped).

TWA

This instruction forms the negative of the count for the loop counter.

STA COUNT

Initializes the loop counter

LDX = A105

This instruction initializes the index register to point to the first location to be cleared.

CLA

Clears the A-register for storing into the locations to be cleared.

LOOP STA 0,1

Clears the location pointed to by the index register

DNX 1

Decrements the index register by 1. Each time through the loop, the index register points to the next location to be cleared.

INC COUNT

Increments the loop counter by 1, making it less negative.

JMP LOOP

When the loop counter reaches 0, this instruction is skipped and execution resumes at DONE. Otherwise, the computer will go back to LOOP for its next instruction.

DATA GENERATING PSEUDO–OPERATIONS

The pseudo-operations described in this section are:
DC – DATA CONSTANT
PTR – ADDRESS POINTER
TXT – CHARACTER STRING

Because the syntax for expressions in the LEAP language is not ambiguous, only two pseudo-operations (DC and TXT) are required for generating data values within a program. It should be noted that data of different types (decimal, hexadecimal, USASCII, address) can be generated from one source line; it is not necessary to have a different DC statement for each of the different types. Up to 7 subfields per source line can be written. The maximum number or words reserved with one DC statement is 28.

It is sometimes desirable to have what may appear to be superfluous operations; i.e., two different operations may be used to generate the same result. This is the case with the DC and PTR pseudo-operations. By using the DC pseudo-operation, it is possible to duplicate any of the capabilities provided with the PTR pseudo-operation. The reason for having the PTR pseudo-operation is to provide a convenient documentation aid. As a courtesy to others who may have to understand a program, it is recommended that the PTR pseudo-operation be used whenever an address or indirect address constant is used. This will make it easy to quickly differentiate between actual data and pointers to memory locations (both of which may be data, but the intended functions are not the same).

DC – Data Constant

LOCATION	OPERATION	VARIABLE
normal	DC [,K]	exp 1, exp 2,..., expn

This pseudo-operation gives the programmer the capability to have LEAP convert certain types of data and expressions into the hexadecimal code used by the MAC 16.

If there is a symbol in the LOCATION field, the value and mode of the symbol will be the value and mode of the LOCATION counter at the time the symbol is encountered.

The optional OPERATION sub-field (,K) is the field size (in words) that will be generated for each value and may be an evaluative absolute expression (no forward or

external references) that results in an integer in the range 1≤K≤4. If K is not in the range 1≤K≤4, the error code "O" will be printed on the listing on the line of the source statement and K will be assumed to be 1.

The LOCATION counter will be incremented by the value of K for each expression in the VARIABLE field of the DC pseudo-operation.

The expressions in the VARIABLE field must follow the rules described in Section 2 — Expressions. Any legal expression may be written in the VARIABLE field. If two or more expressions are written, there must be a comma (,) between the expressions. Up to 7 expressions can be written in the VARIABLE field.

The DC pseudo-operation generates each value in the list into a field whose size is K words (if K is specified) or one word (if K is not specified).

When the field size to be generated for each value is one word, the expressions in the value list must be evaluated as one of the following:

1. Single precision fixed-point decimal data.
2. Hexadecimal values of from one to four hexadecimal digits. If fewer than four hexadecimal digits are written, the digits are right justified in a data word with leading hexadecimal zeros instead. If more than four digits are written, the last four are entered in a data word and the remaining digits are truncated.

```
                                0001 *    EXAMPLE OF DC PSEUDO-OPERATION
     0000 R 0218                0002 EX1     DC      536,-22,1,$12345,'SD','S'
     0001 R FFEA
     0002 R 0001
F    0003 R 0001
     0004 R D3C4
     0005 R 00D3
     0006 R 0000                0003 EX2     DC,2    536,-22,1,$12345,'LEC'
     0007 R 0218
     0008 R FFFF
     0009 R FFEA
     000A R 0000
     000B R 0001
     000C R 0001
     000D R 2345
C    000E R 0000
     000F R C5C3
E    0010 R 0042                0004 EX3     DC      $42R
            0010                0005 START   EQUR    *-1
E    0011 R 0000                0006 EX4     DC,2    ,START,,3.14B5,3.14
     0012 R 0000
     0013 R 0000
     0014 R 0010
E    0015 R 0000
     0016 R 0000
     0017 R 0000
     0018 R 0C8F
     0019 R 4132
     001A R 3D70
            0000                0007         END
*1
OK
```

Example:

Value	Dataword
$ ABC	0ABC
$ 12345	2345

3. USASCII text of one or two characters. A 2-character string fills a word. A single character is placed in the right byte of a word and zeros are placed in the left byte. If a character string contains more than two characters, an error is noted and the last two characters are entered in the data word.
4. A symbol. The value of the symbol becomes the data entry.

When K=2, double precision fixed point data; single precision floating point data; hexadecimal values of up to eight digits and character strings of up to four characters may be used.

When K=4, double precision floating point data; hexadecimal values of up to sixteen digits and character strings of up to eight characters are allowed.

Examples of the DC pseudo-operation follow.

Note that in the example listing, there is an error code "C" to the left of each incorrectly converted line. In Ex. 1, $ 12345 is incorrect because it takes more than a single word to convert it. In Ex. 3, $42R is incorrect because 'R' is not a legitimate hexadecimal character.

PROGRAMMER | DATE

PROGRAM

LOCATION	OPERATION	VARIABLE	COMMENTS
1 2 3 4 5 6	7 8 9 10 11 12 13	14 15 20 25	32 35 40 45 50 55
* EXAMPLE ØF	DC PSEUDØ-ØPERATIØN		
EX1	DC	536,-22,1,$12345,'SD','S'	
EX2	DC,2	536,-22,1,$12345,'LEC'	
EX3	DC	$42R	
START	EQUR	*-1	
EX4	DC,2	,START,,3.14B5,3.14	
	END		

PTR — Address Pointer

LOCATION	OPERATION	VARIABLE
normal	PTR	exp

This pseudo-operation gives the programmer the capability to generate a single word of data. Although there is no restriction to the type of single-word data that can be generated, it is recommended that the use of the PTR pseudo-operation be limited to generating address or indirect address words.

TXT — CHARACTER String

LOCATION	OPERATION	VARIABLE
normal	TXT, MM	Character String

This pseudo-operation enables the user to assemble a character string for use as data.

The character string is a USASCII text string as described in Section 2. The character string is assembled in binary-coded form, two characters per word. A blank is inserted as the second character of the last word if the number of characters is odd. The string can contain up to 58 ASCII characters per source statement and starts after one blank character following MM.

Example of TXT pseudo-operation:
TXT, 11 VALUE OF XX

Generates 6 words as follows:

V	A
L	U
E	
O	F
	X
X	

The pseudo-operation mnemonic (TXT), the subfield delimiter (comma), and the character count (MM) must be wholly contained within the operation field. Blanks within this field result in a misinterpretation of the statement and improper object code is generated.

LISTING CONTROLLING PSEUDO—OPERATIONS

The following paragraphs describe those pseudo-operations provided in the LEAP language to give the programmer control over the documentation of a program. None of the pseudo-operations in this section have any effect on the object code produced by LEAP. The pseudo-operations described in this section are:

EJECT — Skip to the top of the next page*
LIST — List during Pass 2
NLIST — Don't list during Pass 2
SPACE — Space N lines before resuming listing*
TITLE — Heading to be printed at top of page*
LSTSY — List symbol table in alphabetical order

*These pseudo-operations are not listed unless the pseudo-operation is stated.

EJECT — Skip To The Top Of The Next Page

LOCATION	OPERATION	VARIABLE
normal	EJECT	ignored

This pseudo-operation gives the programmer the capability to specify to LEAP that a new page of printed output should be started before resuming the program listing.

When LEAP encounters an EJECT pseudo-operation, the following action takes place:

1. The paper in the listing output device is spaced to the start of the next page.
2. The heading (if any) and page number are printed at the top of the page.
3. The paper is spaced down four lines.
4. The listing resumes.

LIST — List During Pass 2

LOCATION	OPERATION	VARIABLE
normal	LIST	ignored

This pseudo-operation gives the programmer the capability to specify to LEAP that the program listing should be listed on the listing device during pass 2. The LIST pseudo-operation is printed on the listing.

When the LIST pseudo-operation is in effect, all skipped codes are listed (see SKIPF and SKIPT).

The program listing will continue to be produced until the NLIST pseudo-operation is encountered. If neither the LIST nor NLIST pseudo-operations are encountered, LEAP lists the program on the listing device (see Appendix C).

NLIST – Don't List During Pass 2

LOCATION	OPERATION	VARIABLE
normal	NLIST	ignored

This pseudo-operation gives the programmer the capability to specify to LEAP that the program listing should not be listed on the listing device (during pass 2). The NLIST pseudo-operation is printed on the listing.

The program listing will continue to be inhibited until a LIST pseudo-operation is encountered.

SPACE – Space n Lines Before Resuming Listing

LOCATION	OPERATION	VARIABLE
normal	SPACE	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that the number of lines specified by the expression in the VARIABLE field should be spaced up before resuming the program listing. The SPACE pseudo-operation is not printed on the listing unless the LIST pseudo-operation is in effect.

The expression in the VARIABLE field must be absolute. If the expression is not absolute, the error code "E" is printed on the listing, along with the SPACE pseudo-operation.

If the listing has been inhibited (NLIST), the SPACE pseudo-operation has no effect.

If the value of the expression in the VARIABLE field is large enough to cause spacing beyond the bottom of the current page, spacing is terminated, the action described for the EJECT pseudo-operation is taken, and listing resumes in the normal manner.

TITLE – Heading to be Printed at top of Page

LOCATION	OPERATION	VARIABLE
normal	TITLE	'character string'

This pseudo-operation gives the programmer the capability to specify a heading to be printed on the program listing. The TITLE pseudo-operation is not printed on the listing unless the LIST pseudo-operation is in effect.

When LEAP encounters a TITLE pseudo-operation, the following action takes place:

1. All of the characters in the character string are placed in the heading buffer.
2. The action described for the EJECT pseudo-operation takes place.

The contents of the heading buffer continues to be printed on each page until another TITLE pseudo-operation is encountered.

If printing is inhibited (NLIST pseudo-operation), printing does not occur, but the characters in the character string is still placed in the heading buffer.

If no TITLE pseudo-operation is encountered in the source program, the contents of the heading buffer encompass all spaces.

LSTSY – List Symbol Table

LOCATION	OPERATION	VARIABLE
normal	LSTSY	ignored

This pseudo-operation gives the programmer the capability to specify to LEAP that the assembly symbol table should be listed on the listing device (after pass 2) in alphabetical order. The LSTSY pseudo-operation is printed on the listing.

When LEAP encounters a LSTSY pseudo-operation, the following action takes place:

1. The line in which the LSTSY pseudo-operation appears is printed on the listing (subject to NLIST and LIST pseudo-operations).

2. An internal flag is set to denote that the LSTSY pseudo-operation is encountered.
3. When the END pseudo-operation is encountered, the Symbol (name or label) table is printed in alphabetical order along with the value of each of the symbols.

EXTERNAL REFERENCE PSEUDO–OPERATIONS

It is desirable to segment programs to decrease the size of source programs. When this is done, it is usually necessary to reference elements that are in another segment from the one being assembled. The pseudo-operations described in this section make it possible for the programmer to conveniently do this.

There are two pseudo-operations provided in the LEAP language for external reference and definition. If either (or both) of these pseudo-operations are used in a source program, the Extended Loader must be used to load the object code produced by the assembly process. The two pseudo-operations are:

EXTRN – EXTERNAL SYMBOL (DEFINED OUTSIDE THIS PROGRAM) REFERENCE

ENTRY – EXTERNAL SYMBOL DEFINITION

EXTRN – External Symbol

LOCATION	OPERATION	VARIABLE
normal	EXTRN	symbol, symbol, ...,symbol

This pseudo-operation gives the programmer the capability to specify to LEAP that the symbol (s) listed in the VARIABLE field are external symbols (not defined within the program). Programs which reference external symbols must be loaded by the Extended Loader.

There must be at least one symbol in the VARIABLE field; there can be as many as 7 symbols (separated with commas) in the VARIABLE field. If no symbol is encountered in the VARIABLE field, the error code "V" is printed on the listing.

The most common use for the EXTRN pseudo-operation is to specify the names of subroutines referenced within a program, but are assembled separately. The EXTRN pseudo-operation causes the proper codes to be produced in the object code so that the Extended Loader is able to complete linkages to subroutines.

Although the use of the EXTRN pseudo-operation is primarily associated with subroutine reference, it is not restricted to this. It is often desirable to reference an element or group of elements (array) that is external to the program. This can also be done by using the EXTRN pseudo-operation.

Symbols in the VARIABLE field of an EXTRN statement may not appear as statement names in the same program as the EXTRN statement itself. These will be flagged with an error code E. The statement with the name which appears in the EXTRN statement VARIABLE field is flagged with an error code S.

ENTRY – External Symbol Definition

LOCATION	OPERATION	VARIABLE
normal	ENTRY	symbol, symbol, ...,symbol

This pseudo-operation gives the programmer the capability to specify to LEAP that the symbols appearing in the VARIABLE field may be referenced externally (by other programs).

There must be at least one symbol and, there can be as many as 7 symbols (separated with commas) in the VARIABLE field. If no symbol appears in the VARIABLE field, the error code "V" is printed on the listing.

The example below is a subroutine that refers to another subroutine; therefore, both the ENTRY and EXTRN pseudo-operations are used.

The subroutine is referenced externally as COSINE. Since the subroutine is "calling" another subroutine which is external, the EXTRN pseudo-operation is used to define the external subroutine. If the EXTRN pseudo-operation is included, the reference to SINE would have the error code "U" printed on the listing (the symbol would have been undefined).

```
PROGRAMMER                                                    DATE
PROGRAM
LOCATION      OPERATION      VARIABLE                                   COMMENTS
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15    20        25          32   35   40   45   50   55
* EXTERNAL SYMBOL EXAMPLE
          ENTRY     COSINE
COSINE    ADD       =1.62
          STX       TEMP
          JMM       SINE
          LDX       TEMP
          JMX       0
          EXTRN     SINE
TEMP      DS        1
          END
```

STORAGE ALLOCATING PSEUDO–OPERATIONS

When it is necessary to reserve an area of one or more words for temporary or calculated values, LEAP provides the capability of specifying that area (of any size) with just one source statement. This prevents forcing the programmer to write a line of code for each word that is to be reserved. Three pseudo-operations are described in the following paragraphs:

CLEAR – CLEAR AND RESERVE AN AREA
COMN – RESERVE AN AREA IN COMMON
DS – RESERVE A DATA STORAGE AREA

CLEAR – Clear and Reserve an Area

LOCATION	OPERATION	VARIABLE
normal	CLEAR	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that an area of memory should be reserved and cleared (set to zeros) at load time. The CLEAR pseudo-operation is basically the same as the DS pseudo-operation, except that the reserved area is cleared (the DS pseudo-operation does not clear the reserved area).

If a symbol is encountered in the LOCATION field, the value of the symbol will be the location of the first word of the area to be reserved and cleared. References to the other words in the area must be relative to the first word.

The value of the expression in the VARIABLE field is the number of words to be reserved and cleared. The value of the expression must be absolute, and any symbols used in the expression must have been previously defined (no forward reference). If the value of the expression is not absolute, or if a symbol is used that has not been previously defined, the error code "P" will be printed on the listing with the line. Also, if nothing is entered for the expression, the error code "V" will be printed.

When LEAP encounters a CLEAR pseudo-operation, the following actions take place:

1. If there is a symbol in the LOCATION field, the symbol will be assigned the mode and value of the LOCATION counter.
2. The expression in the VARIABLE field will be evaluated.
3. The value of the expression in the VARIABLE field will be added to the LOCATION counter.

4. Object code is produced to specify to the Extended Loader the number of words that should be cleared at load time.

COMN — Reserve an Area in COMMON

LOCATION	OPERATION	VARIABLE
normal	COMN	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that an area of memory should be reserved in the COMMON area.

If a symbol is encountered in the LOCATION field, the value of the symbol will be the location of the (numerically) first word of the area to be reserved. References to the other words in the area must be defined symbolically.

The value of the expression in the VARIABLE field is the number of words to be reserved. The value of the expression must be absolute, and any symbols used in the expression must have been previously defined (no forward reference). If the value of the expression is not absolute, or if a symbol is used that has not been previously defined, the error code "P" will be printed on the listing with the line. Also, if nothing is entered for the expression, the error code "V" will be printed.

When LEAP encounters a COMN pseudo-operation, the following action takes place:

1. If there is a symbol in the LOCATION field, it will be assigned the current value of the COMMON counter (the COMMON counter is initialized to zero at the start of an assembly).

2. The expression in the VARIABLE field will be evaluated and the resultant value will be added to the COMMON counter.

The Extended Loader assigns the external location of COMMON in memory at load time. The value assigned to the symbol in the LOCATION field of a COMN pseudo-operation is actually the value of the offset from the start of COMMON.

DS — Reserve Data Storage Area

LOCATION	OPERATION	VARIABLE
normal	DS[,K]	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that an area of memory should be reserved at load time. The DS pseudo-operation is the same as the CLEAR pseudo-operation, except that the reserved area is not cleared.

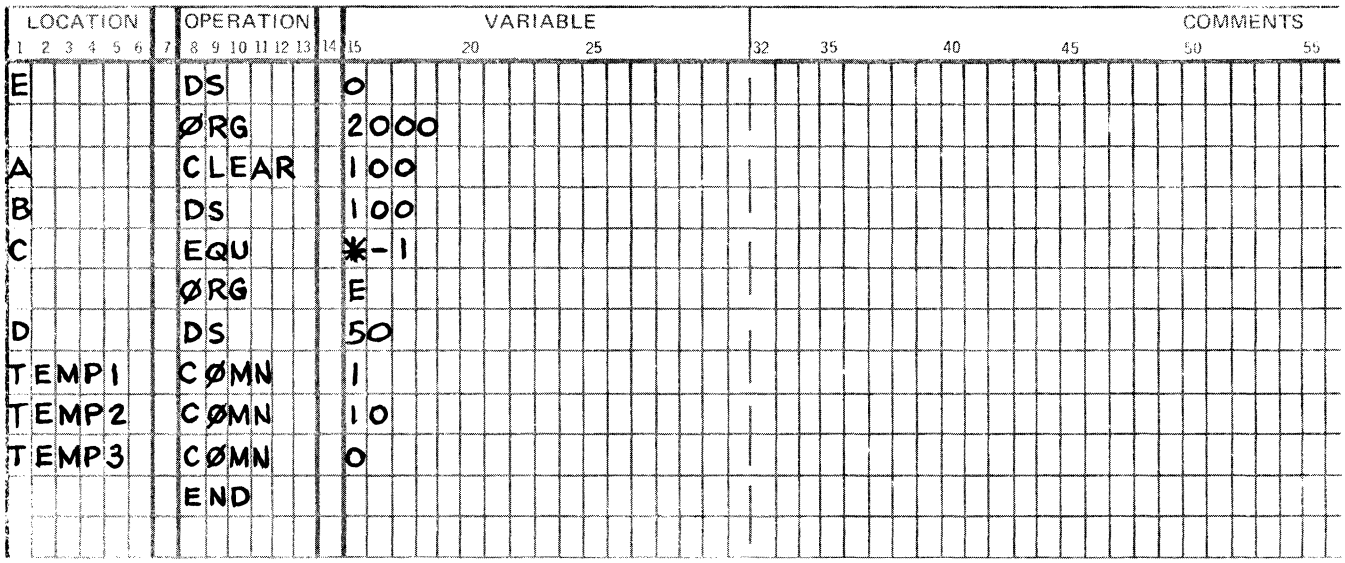
If a symbol is encountered in the LOCATION field, the value of the symbol will be the location of the first word of the area to be reserved. References to the other words in the area must be relative to the first word.

The optional OPERATION sub-field [,K] is the field size (in words) that will be generated for the expression in the VARIABLE field and may be a evaluatable absolute expression (no forward or external references) that results in an integer in the range $1 \leq K \leq 4$. If K is not specified, it will be assumed as 1 (default condition). If K is not within the limits of from 1 to 4, the error code O will be printed on the listing on the line with the source statement and K will be assumed to be 1.

The value of the expression in the VARIABLE field is the number of fields to be reserved. The value of the expression must be absolute, and any symbols used in the expression must have been previously defined (no forward reference). If the value of the expression is not absolute, or if a symbol is used that has not been previously defined, the error code "P" will be printed on the listing with the line. Also, if nothing is entered for the expression, the error code "P" will be printed.

When LEAP encounters a DS pseudo-operation, the following actions take place:

1. If there is a symbol in the LOCATION field, the symbol will be assigned the mode and value of the LOCATION counter.
2. The value of K will be recorded.
3. The expression in the VARIABLE field will be evaluated.
4. The value of the expression in the VARIABLE field will be multiplied by K and added to the LOCATION counter.



In this example, the following action takes place:

1. Initial values:
   A) LOCATION counter set to zero
   B) COMMON counter set to zero
   C) relocatable mode set
   D) the object is defined as extended format.
2. The first line defines E as relocatable 0 (default condition is relocatable)
3. The second line sets the LOCATION counter to 07D0 (hex), absolute
4. The symbol A is assigned the absolute mode with a value of 07D0 (hex)
5. The LOCATION counter is increased to 0834 (hex) and object code is produced to tell the extended loader to clear from 07D0 to 0833 (hex)
6. The symbol B is assigned the absolute mode with a value of 0834 (hex)
7. The LOCATION counter is increased to 0898 (hex)
8. The symbol C is assigned the absolute mode with a value of 0897 (hex)
9. The LOCATION counter is set to relocatable 0
10. The symbol D is assigned the relocatable mode with a value of 0000
11. The LOCATION counter is increased to 0032 (hex)
12. The symbol TEMP1 is assigned the value of the COMMON counter (0000)
13. The COMMON counter is increased to 0001 (hex)
14. The symbol TEMP2 is assigned the value of the COMMON counter (0001)
15. The COMMON counter is increased to 000B (hex)
16. The symbol TEMP3 is assigned the value of the COMMON counter (000B)

SYMBOL DEFINING PSEUDO–OPERATIONS

The pseudo-operations described in this section make it possible for the programmer to assign values (and mode) to symbols without necessarily relying on the value of the LOCATION counter. The pseudo-operations discussed are:

EQU – SYMBOL EQUALS VALUE AND MODE OF EXPRESSED

EQUR – SYMBOL EQUALS VALUE OF EXPRESSION, RELOCATABLE MODE

REDEF – REDEFINE SYMBOL

EQU – Symbol Equals Value and Type of Expression

LOCATION	OPERATION	VARIABLE
symbol	EQU	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that the symbol appearing in the LOCATION field is to be assigned the value and mode of the expression in the VARIABLE field.

If a symbol is not entered in the LOCATION field, the error code "S" is printed on the listing. If no expression is entered (left blank), the error code "V" is printed on the listing. Any symbols used in the VARIABLE field must be previously defined; if not, the error code "P" will be printed on the listing.

EQUR – Symbol Equals Value of Expression, Relocatable Mode

LOCATION	OPERATION	VARIABLE
symbol	EQUR	expression

This pseudo-operation gives the programmer the capability to specify to LEAP that the symbol appearing in the LOCATION field is to be assigned the value of the expression in the VARIABLE field, but that the mode assignment is relocatable.

If a symbol is not entered in the LOCATION field, the error code "S" is printed on the listing. If no expression is entered (left blank), the error code "V" is printed on the listing. Any symbols used in the VARIABLE field must be previously defined; if not, the error code "P" will be printed on the listing.

REDEF – Redefine Symbol

LOCATION	OPERATION	VARIABLE
normal	REDEF	Symbol, expression

The REDEF pseudo-operation, like EQU, enables the user to define a symbol by assigning it the value and mode of the expression in VARIABLE field.

The REDEF pseudo-operation differs from the EQU pseudo-operation in that any symbol defined by a REDEF is redefined later by means of a subsequent REDEF.

If a symbol defined via a REDEF pseudo-operation is redefined but the user writes an EQU instead of a new REDEF pseudo-operation, LEAP prints an "M" error code and retains the earlier definition.

If symbol is not entered in the VARIABLE field, the error code "S" is printed on the listing. If no expression is entered (left blank), the error code "V" is printed on the listing. Any symbols used in the variable field must be previously defined; if not, the error code "P" will be printed on the listing.

MISCELLANEOUS PSEUDO–OPERATIONS

The pseudo-operations discussed in the section are:
SETB – SET THE LOADING B FLIP-FLOP
RESB – RESET THE LOADING B FLIP-FLOP
PRINT – PRINT DURING PASS 1

SETB – Set the Loading B Flip-Flop

LOCATION	OPERATION	VARIABLE
normal	SETB	ignored

The SETB pseudo-operation enables the programmer to specify to LEAP that the status of the B flip-flop is a "one".

1. SETB generates type code 16 (see Section 5), which is a directive to the Extended Loader indicating that automatic depaging should be through the fixed base page (location 512_{10}-1023_{10}).
2. If LEAP is assembling object code in bootstrap format, SETB is used to specify to LEAP that the correct base page of the program is page 1 (locations 512_{10}-1023_{10}). Any memory reference instruction must reference a location either in the local page or to page one. Otherwise, the error code "B" appears on the listed source line.

If neither SETB nor RESB appears at the beginning of the program, LEAP assumes that the B flip-flop is a "1" (initial or default condition).

The extended loader's B flip-flop can also be set via the SEX machine instruction. This instruction acts as a pseudo-operation also, and generates a type code 16 on the object tape or control error detection.

RESB – Reset the Loading B Flip-Flop

LOCATION	OPERATION	VARIABLE
normal	RESB	ignored

The RESB instruction enables the programmer to specify to LEAP that the status of the B flip-flop is a "zero".

1. RESB generates type code 17 (see Section 5) which is a directive to the Extended Loader indicating that automatic depaging should be through the floating base page (refer to MAC 16 Programmer's Manual).
2. If LEAP is assembling object code in bootstrap format, RESB is used to specify to LEAP that the correct base page of the program is relative to the current LOCATION Counter value (refer to page control in the MAC 16 Programmer's Manual). Any memory reference instruction must reference a location either in the local page or to the correct floating base page. Otherwise, the error code "B" will appear on the listed source line. If neither SETB nor RESB appear at the beginning of the program, LEAP will assume that the B flip-flop is a "1" (initial or default condition).

The extended loader's B flip-flop can also be reset via the REX machine instruction. This instruction acts as a pseudo-operation also, and it generates a type code 17 on the object tape.

PRINT – PRINT During Pass 1

LOCATION	OPERATION	VARIABLE
normal	PRINT	'character string'

This pseudo-operation gives the programmer the capability to have the remarks specified by the USACII character string printed on the systems teleprinter during pass 1. No object code is generated by the PRINT pseudo-operation.

The characters in the character string of the source line are printed on the teleprinter as they are; no scan takes place. Assembling continues with the next source line in the normal manner.

MACRO Definition

A macro is a group of one or more statements inserted whenever a corresponding macro reference or call occurs. Macros are, in a sense, analogous to subroutines which can be called by a reference or calling statement. Like a subroutine, a macro has a name by which it is called. Macros are generally used as an aid for programming in-line standard functions in which the sequence of operations (instructions) does not vary, but the expressions in the VARIABLE fields (parameters) do. This is illustrated in Example 1 which defines a double precision, fixed point add. (The contents of location D and D+1 are added to the contents of location E and E+1. The result is stored in locations F and F+1. HALT is the location of the overflow error routine.)

A macro definition must precede its first calling statement. A macro is defined when LEAP encounters the special character # in the first column of a location field. This is the macro header. All statements within the macro definition are called model statements and, except for the last statement are terminated with the special continuation character (;). The collection of model statements for a macro comprise the "macro skeleton". The last statement signals the end of the macro skeleton (by failing to have the continuation character).

Parameters

The parameters of a macro are the operands of the reference or calling statement. Within a macro definition, the parameters are represented by the special character pound sign (#) followed by either a number or an asterisk followed by a number e.g. #2 or #*2. Since macros are positional rather than key-word, #1 refers to the first parameter on the reference statement, #2 refers to the second parameter on the reference statement, etc. #*1 refers to the indirect attribute associated with the first parameter on the reference statement, #*2 refers to the indirect attribute associated with the second parameter on the reference statement, etc. In a model statement, a reference to a parameter may be written anywhere an item of an expression can be written. However, parameter references are meaningful only within the VARIABLE or OPERAND field of the model statement.

The values of the parameters will normally be different each time the macro is called or referenced. Macro definitions and references need not agree in number of parameters. However, parameters not defined on the reference line will have the value zero. The error code "A" will appear on the listing if no parameters appear on the reference line.

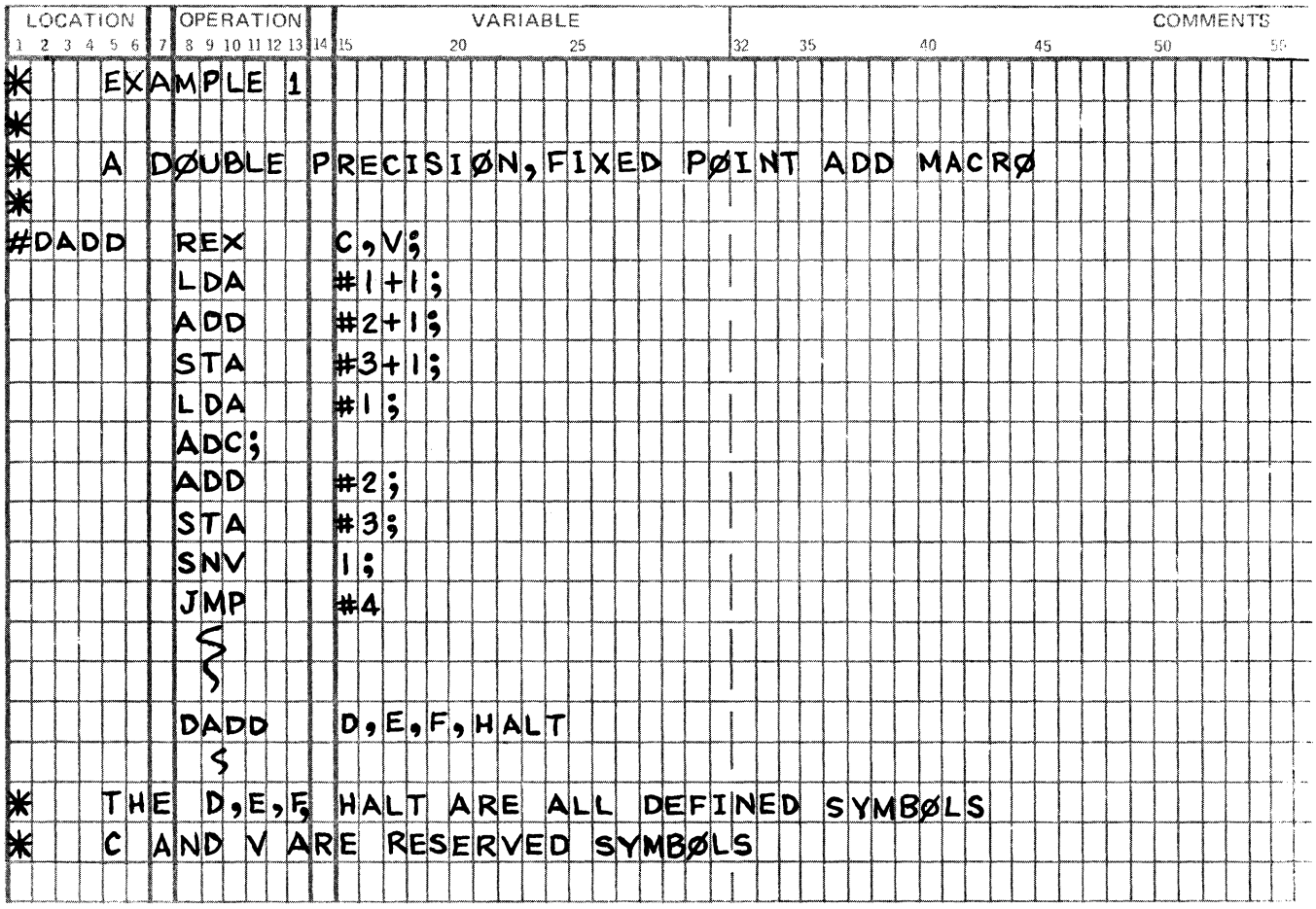
A macro is referenced or called by writing the name of the macro in the OPERATION field of a statement. The VARIABLE field of the calling statement will contain the

parameters to the macro. If more than one parameter is to be supplied to the macro, the parameters are written one after the other, separated by commas. The last parameter is not followed with a comma. The maximum number of parameters capable of being passed to the macro is seven.

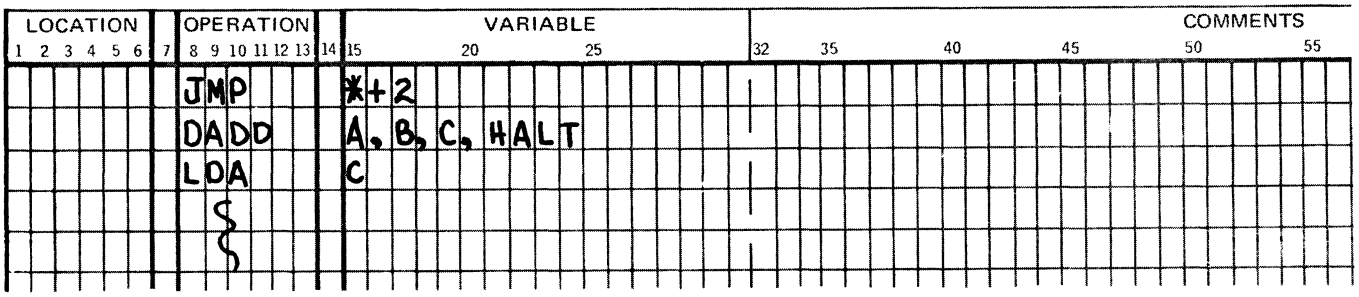
Parameter substitution is positional rather than keyword, i.e., the first parameter is equated to (#1) in the macro expansion, the second parameter is equated to #2', etc. Other symbols besides those used as parameters in macro calls may also be included in macro definitions. In addition, the reserved symbols C,V,R,S,B, and H may be used in a macro as in example 1, where C and V are used.

In this example, argument substitutions are made. When the macro name DADD is encountered in the OPERATION field, the object code corresponding to the following source lines is inserted in its place:

```
REX     C, V
LDA     D + 1
ADD     E + 1
STA     F + 1
LDA     D
ADC
ADD     E
STA     F
SNV     1
JMP     HALT
```



If the following code had been written, using the above macro definition:



The JMP *+2 instruction would cause a jump to the second word of the macro definition, not the source line following the DADD operation. For this reason, it is recommended that relative notation not be used preceding macro calls, especially, if the programmer has macro definitions that generate more than one instruction.

This macro uses SKIPT and SKIPF. When the MOVE macro is called by:

MOVE 1, ALPHA, BETA

the following coding is generated:

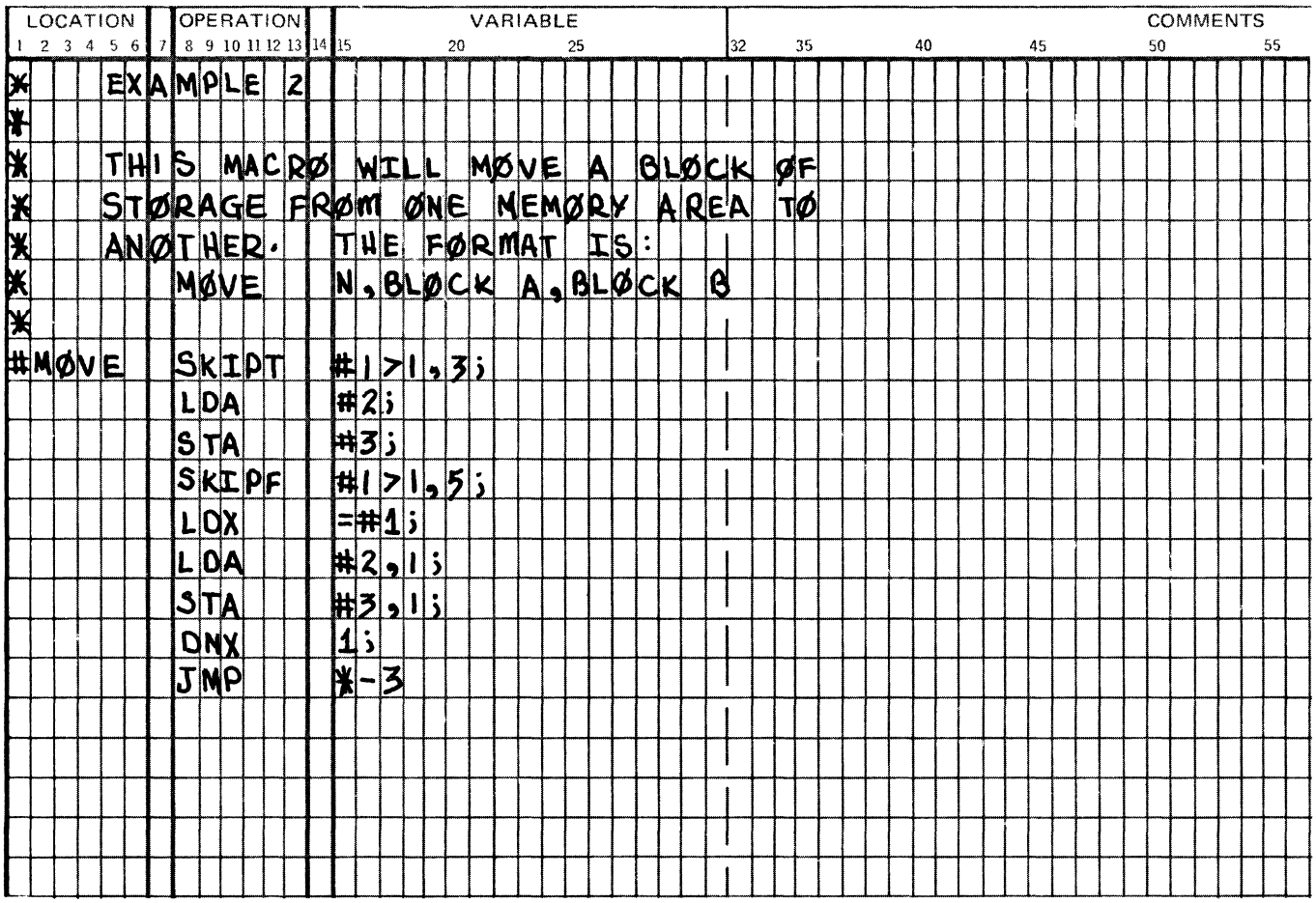
```
LDA     ALPHA
STA     BETA
```

When the MOVE MACRO IS CALLED BY;

MOVE 5, ALPHA, BETA

the following coding is generated:

```
LDX     =5
LDA     ALPHA, 1
STA     BETA, 1
DNX
JMP     * -3
```



Example 3 illustrates a technique of microcoding the MAC 16 through LEAP. When LEAP encounters "TAL14" in the OPERATION field of a source line, it generates object code of type 1 (see Section 5) consisting of 0589. This particular instruction could have been generated by "TAL9". The purpose of generating the new operation code is for ease in documentation. This particular instruction will logically OR the contents of the A register into register L1 and L4.

PROGRAMMER DATE

PROGRAM

LOCATION	OPERATION	VARIABLE	COMMENTS
1 2 3 4 5 6	7 8 9 10 11 12 13	14 15 20 25	32 35 40 45 50 55
* EXAMPLE	3		
*			
* THIS MACRØ DEFINES AN ØPERATIØN CØDE			
*			
#TAL14	DC	$0589	

Section 4

Assembling MAC 16 Instructions

This section contains information on the assembly process performed by LEAP when the mnemonic for a MAC 16 instruction is encountered in the OPERATION field. The process performed by LEAP varies with the type of instruction encountered. The six classifications of MAC 16 instructions are:

CLASS 0: 16 BIT, UNMODIFIED

No entry is allowed in the VARIABLE field on this class of instructions. If any entry is encountered, the error code "V" is printed on the listing. The following instructions are in Class 0:

ABA – Absolute Value of A
ADC – Add Carry to A
CLA – Clear A
HLT – Halt
JMA – Jump Direct Thru A
LSB – Load A (0-5) with status of C,V,R,S,B and H
NOP – No Operation
ONA – One's Complement of A
SSB – Store A^{0-5} into status bits C,V,R,S,B and H
TLA – Transfer Level to A
TSA – Transfer Switches to A
TWA – Two's Complement of A
XXA – Exchange Index and A
MPY – Multiply A
DIV – Divide A

CLASS 1: MEMORY REFERENCE

This class of instructions is the only class that allows indirect addressing and indexing. The first subfield in the VARIABLE field is evaluated and a 16-bit address is produced, if in the extended format; if in the bootstrap format, the address is checked for reference outside the current page or the base page. In the bootstrap format, any reference outside the current page causes the error code "B" to be printed on the listing.

The address subfield is terminated with either a blank character or a comma (,). If a comma is encountered, the next subfield is evaluated for indexing. The value of the index subfield is treated modulo 1. If nothing is entered in the address subfield, the error code "A" is printed on the listing.

The following instructions are in Class 1:

ADD – Add Memory to A
ANA – AND memory with A
CAA – Compare A with Memory Arithmetically
INC – Increment Memory and Skip if Zero
JMM – Jump and Mark
JMP – Jump
JRL – Jump and Reset Level
LDA – Load A from Memory
LDX – Load Index from Memory
ORA – OR from Memory to A
STA – Store A in Memory
STL – Store Left Byte of A in Memory
STR – Store Right Byte of A in Memory
STX – Store Index in Memory
SUB – Subtract Memory from A

CLASS 2: I/O INSTRUCTIONS

This class of instructions requires a subfield for both the M and N fields. If either the M or N subfield is omitted, the error code "V" will be printed on the listing. Values in either subfield are evaluated modulo 16. If either of the values is too large (more than 4 bits), the error code "F" is printed on the listing. The M subfield is equivalent to the address subfield of a memory reference instruction, and the N subfield is equivalent to the index subfield.

The following instructions are in Class 2:

EAI – External Address In
ECO – External Command Out
EDI – External Data In
EDO – External Data Out
ESI – External Status In

CLASS 3: SKIP INSTRUCTIONS

This class requires an entry in the VARIABLE field. If nothing is encountered, the error code "V" is printed on the listing. The VARIABLE field is evaluated in the following manner:

1. If the value in the VARIABLE field is in the range of 0 to 15 inclusive and the mode is absolute, the value is inserted directly into the N field of the instruction.
2. If the value in the VARIABLE field is greater than 15 or the mode is relocatable, it is assumed to be a memory address. LEAP computes the N value required to skip to the specified memory location (forward only). If the resulting value is larger than 4 bits, the error code "F" is printed on the listing; the value is truncated to 4 bits and inserted into the instruction's N field.

The following instructions are in Class 3:

SAG – Skip if A is Greater than Zero
SAN – Skip if A is Negative
SAZ – Skip if A is Zero
SKN – Skip if A is Normalized
SKP – Skip Unconditional
SKX – Skip if Index is Zero
SLZ – Skip if Least-Significant Bit of A is Zero
SNB – Skip if Base Page Control is Zero
SNC – Skip if C is Zero
SNH – Skip in Interrupt Not Inhibited
SNV – Skip if V is Zero
SNR – Skip if R is Zero
SNS – Skip if S is Zero

CLASS 4: N FIELD

This class requires an entry in the VARIABLE field. If nothing is encountered, the error code "V" is printed on the listing. The VARIABLE field is evaluated modulo 16; if the value is greater than 15 (more than 4 bits), the error code "F" is printed on the listing. The following instructions are in Class 4:

ALI – Arithmetic Left Shift & Insert
ALS – Arithmetic Left Shift
ARI – Arithmetic Right Shift & Insert
ARS – Arithmetic Right Shift
DNX – Decrement Index and Skip on Zero
INX – Increment Index and Skip on Zero
JIX – Jump Indirect Thru Index
JMX – Jump Direct Thru Index
LAX – Load A Direct from Memory Location Specified by Index
LIX – Load A Indirect from Memory Location Specified by Index
LLC – Logical Left Shift, Closed
LLI – Logical Left Shift & Insert
LLN – Logical Left Shift, No Change to C
LLO – Logical Left Shift, Open
LRC – Logical Right Shift, Closed
LRI – Logical Right Shift & Insert
LRN – Logical Right Shift, No Change to C
LRO – Logical Right Shift, Open
SAX – Store A in Location Specified by Index
SIX – Store A Indirect thru Location Specified by Index
TAL – Transfer A to L Register

CLASS 5: IMMEDIATES

This class of instructions requires an entry in the VARIABLE field. If nothing is encountered, the error code "V" is printed on the listing. The VARIABLE field is evaluated modulo 256; if the value is greater than 255 (more than 8 bits), the error code "F" is printed on the listing.

The following instructions are in Class 5:

ADI – Add Immediate to A
LDI – Load A Immediate
SBI – Subtract Immediate from A

CLASS 6: STATUS MODIFIERS

This class of instructions requires an entry in the VARIABLE field. If nothing is encountered, the error code "V" is printed on the listing. The VARIABLE field is evaluated modulo 64; if the value is greater than 63 (more than 6 bits), the error code "F" is printed on the listing.

The following instructions are in Class 6:

SEX – Set Sense Indicator
REX – Reset Sense Indicator

For these instructions, the indicators may be represented by any combination of the reserved symbols C,V,R,S,B and H. When writing the indicators on a coding sheet, they are separated by commas (,).

Section 5 Input/Output

SOURCE PROGRAM PREPARATION

If cards are not used for the source program, prepare a paper tape according to the following instructions:

Preamble-Start-Of-File

The preamble character ($81) must be used. It is formed (on TTY) by simultaneously depressing the CTRL and A keys.

Card Images

Each image is ended by a carriage-return, line-feed. (See page 2-1.)

End-Of-File

Each program must be terminated by an END record generated by typing the symbol END preceded by one or more spaces and followed by a RETURN and LINE FEED.

ASSEMBLY LISTING FORMAT

This section describes the format of the program listing that is part of the LEAP output during pass 2. The format is discussed in terms of a line printer record with 108 character-positions, plus a carriage control character. LEAP presents a 56-word image to the listing output routine. It is left to the output routine to format the image for the appropriate device; if the output device is the systems teleprinter the teleprinter output routine truncates the image to a 72-character image, plus the carriage-control character.

Page Format

This listing is printed on continuous form paper. If the teleprinter is being used, the output routine keeps a line count so that spacing is in 11 inch increments. The following are the specifications for the page format. There are a total of 66 lines on an 11-inch page (6 lines per inch).

LINES	CONTENTS
1-6	Blank
7	Page Heading (generated by TITLE pseudo-op)
8-11	Blank
12-60	Listing (line formats described below).
61-66	Blank

Line Format

The first word of the 109-character image contains a single character (in USASCII code) which is used by the output routine for controlling the spacing of paper in the listing device. The characters have the following meanings (USASII Standard).

CHARACTER	MEANING
blank	single space before printing.
0	double space before printing.
1	advance to next page, then print.
+	no space (the default condition).

The remaining 54 words of the 55-word image contains the 108 characters to be printed. The characters are packed two per word and are in USASCII code.

The following lists the format of the listing line.

PRINT POSITIONS	CONTENTS
1-4	Error codes (maximum of four), if less than four, the codes are left-justified in the field.
5	Blank

PRINT POSITIONS	CONTENTS
6-9	Four hexadecimal digits, reserved for printing the contents of the Location Counter. The Location Counter is not printed on comments lines, all-blank lines, or the Type 6&7 value codes (see below).
10	Blank
11	One alphabetic character that depends on the mode of the operand: A – absolute R – relocatable E – extend C – common
12	Blank
13-19	The object code; there are five different formats within this field, depending on the value code (see below).
20-23	Blank
24-27	A decimal number that is the identification of the source line assigned by LEAP. LEAP increases the line number by one for each source statement encountered. The first statement is 0001, the second is 0002, etc. This number is used for reference when performing a source program edit with the EDIT program.
28	Blank
29-108	The 80-character source statement.

Value Formats (Positions 13-21)

The format of the value is dependent on the type of the value. The following are the various type codes and the format associated with the code (Appendices B & C give the type codes for all operations).

Type 1 – 16-bit value

PRINT POSITIONS	CONTENTS
13-16	Four hexadecimal digits, representing the 16-bit value.
17-19	Blank

Type 2 – Memory Reference Instruction

PRINT POSITIONS	CONTENTS
13	The hexadecimal operation code.
14	A hexadecimal digit representing the status of X, I, and P bits.
15	Blank
16-19	The 4 hexadecimal digits representing the 16-bit address value of the instruction.

Type 3 – Input/Output Instructions

PRINT POSITIONS	CONTENTS
13-14	The two hexadecimal digits representing the operation code.
15	Blank
16	The hexadecimal digit representing the M field
17	Blank
18	The hexadecimal digit representing the N field

Type 4 – Instructions With N Field

PRINT POSITIONS	CONTENTS
13-15	The three hexadecimal digits representing the operation code.
16	Blank
17	The hexadecimal digit representing the N field

Type 5 – Immediate Instructions

PRINT POSITIONS	CONTENTS
13-14	The two hexadecimal digits representing the operation code.
15	Blank
16-17	The two hexadecimal digits representing the immediate value.

Type 6 – No Value

PRINT POSITIONS	CONTENTS
5-21	Blank

Type 7 – No Location
PRINT

POSITIONS	CONTENTS
13-16	Four hexadecimal digits.

OBJECT CODE

In the discussion that follows, no reference is made to a specific input/output medium. The output device used for producing object code during assembly is of no real concern to LEAP. Section 5 discusses the method used by LEAP to interface with the outside world. The object code is given to the appropriate output routine in a standard format; it is the responsibility of the output routine to prepare the code for the device being driven by the routine. In the same way, the input routine for the loader presents the object code to the loader in a standard format. This section is concerned only with the standard format of the object code.

Two object code formats are produced by LEAP: the bootstrap format and the extended format.

Bootstrap Format

The bootstrap format is absolute and has no capability for automatic de-paging. The Bootstrap Loader is normally used for loading this format. The format of the information produced is (all words are 16-bit words):

First Word

The first word is the 16-bit starting address for loading the record that follows.

Second Word

The second word is in two forms:

If the sign bit (bit 0) is a one, the remainder of the word is ignored and the first word is taken as the starting location for the program. The Bootstrap Loader transfers control to this location and execution continues from there.

If the sign bit (bit 0) is a zero, the remaining 15 bits give the count of the number of words in the record. The count does not include the first word, second word, or checksum word.

Program

The second word is followed by the number of words specified by the second word.

Checksum

The checksum word follows every record.

Extended Format

The extended format is loaded only by the Extended Loader. If external references are made, the Extended Loader is used to handle all external references and definitions.

The object code produced in the extended format is a series of byte-strings of variable length. The first byte of every string is the type code that gives information about the remainder of the string.

The following is a discussion of the type codes produced by LEAP and recognized by the Extended Loader:

Type Code 1: 16-Bit Value, Absolute String
1,N,WORD 1, WORD 2,. . ., WORD N

N = The number of 16-bit words in the string (1 N 255) WORD 1 through WORD N are the 16-bit words to be stored, unmodified; the LOCATION counter is increased by one for each word stored.

Type Code 2: 16-Bit Value, Positive Relocation String:
2,N,WORD 1, WORD 2,. . ., WORD N

N = The number of 16-bit words in the string (1 N 255) WORD 1 through WORD N are the 16-bit words to be stored. Prior to storing, the relocation factor is added to the words. The LOCATION counter is increased by one for each word stored.

Type Code 3: 16-Bit COMMON Reference String:
3, WORD

WORD = The 16-bit number representing offset from the start of COMMON.

Type Code 4: 16-Bit, External Reference String:
4,N,BYTE 1, BYTE 2,. . ., BYTE N,

N = The number of 8-bit bytes used for the external name (0 N 6). If N = 0, the reference is to unlabelled COMMON and no name follows.

BYTE 1 through BYTE N are the 8-bit character codes used for the external name.

Type Code 5: 24-bit Value, Absolute String:
5, OP, WORD

OP is the 4-bit operation code, 3 bits for index, indirect, and page; the least-significant bit is 0 (making a total of 8 bits).

Type Code 6: 24-bit Value, Positive Relocation String:
6, OP, WORD

OP is the 4-bit operation code, 3 bits for index, indirect, and page; the least-significant bit is 0 (making a total of 8 bits).

WORD will be added to the relocation factor; the resultant value will be depaged.

Type Code 7: 24-Bit COMMON Reference String:
7, OP, WORD

OP includes the 4-bit operation code and 3 bits for index, indirect, and page; the least-significant bit is 0 (making a total of 8 bits).

WORD = the 24-bit offset from the start of COMMON.

Type Code 8: 24-Bit Value, External Reference String:
8, N, BYTE 1, BYTE 2,. . ., BYTE N, OP

N = The number of 8-bit bytes used for the external name (0 N 6). If N = 0, the reference is to unlabelled COMMON and no name follows.

BYTE 1 through BYTE N are the 8-bit character codes for the external name.

OP is the 4-bit operation code, 3 bits for index, indirect, and page; the least-significant bit is 0 (making a total of 8 bits).

Type Code 9: END, Absolute String: 9, WORD

WORD is the location to which control will be transferred by the Extended Loader (the starting location of the program just loaded).

Type Code 10: End, Positive Relocation String:
10, WORD

WORD is added to the relocation factor. The resulting value is the location to which control will be transferred by the Extended Loader (the starting location of the program just loaded).

Type Code 11: End, External Value String:
11, N, BYTE 1, BYTE 2,. . ., BYTE N

N = The number of 8-bit bytes used for the external name (1 N 6).

BYTE 1 through BYTE N are the 8-bit character codes for the external name.

Type Code 12: Size of COMMON String:
12, WORD

WORD = The 16-bit number representing the number of locations in COMMON which this program requires.

Type Code 13: Set Absolute Counter String:
13, WORD

WORD is the 16-bit value that will be stored in the Extended Loader's absolute LOCATION counter.

This type code also forces the Extended Loader to use the absolute LOCATION counter for storing values.

Type Code 14: Set Relocatable Counter String:
14, WORD

WORD is a 16-bit value that will be added to the relocation factor and then stored in the Extended Loader's relocatable LOCATION counter.

This type code also forces the Extended Loader to use the relocatable LOCATION counter for storing values.

Type Code 15: Clear String:
15, WORD

WORD is the number of locations to be cleared, starting with the current value of the LOCATION counter. The 16-bit value of WORD is also added to the LOCATION counter.

Type Code 16: SEB String:
16

This type code forces the Extended Loader to depage only through page 1.

Type Code 17: REB String:
17

This type code allows the Extended Loader to depage through the page associated with the LOCATION counter (0 thru 8K, use page 1; 8K thru 16K, use page 2, etc.).

Type Code 18: CHECKSUM STRING:
18, WORD

WORD is the CHECKSUM of all words encountered since the last CHECKSUM.

Type Code 19: Program Size String:
19, WORD

WORD is the number of locations used by the program (not counting depaging).

Type Code 20: Absolute External Definition String:
20, N, BYTE 1, BYTE 2,. . ., BYTE N, WORD

N = The number of 8-bit bytes used for the name (1 N 6). BYTE 1 through BYTE N are the 8-bit character codes for the name. WORD is the value (absolute location) of the name.

Type Code 21: Relocatable External Definition String:
21, N, BYTE 1, BYTE 2,. . ., BYTE N, WORD

N = the number of 8-bit bytes used for the name (1 N 6). BYTE 1 through BYTE N are the 8-bit character codes for the name. Word is the value (relocatable) of the name.

Checksum

The following is the method to be used for forming the checksum (either for the bootstrap format or extended format):
(Initial value of SUM is zero)
LOAD SUM
ADD BYTE
STORE SUM

The checksum will be computed for every byte in the buffer except the checksum itself (includes the type code for the checksum).

APPENDIX A – USASCII CHARACTER SET AND HEXADECIMAL CODES

HEX	CHARACTER
A0	space
A1	!
A2	"
A3	#
A4	$
A5	%
A6	&
A7	' (apostrophe)
A8	(
A9	)
AA	*
AB	+
AC	, (comma)
AD	-
AE	. (period)
AF	/
B0	0
B1	1
B2	2
B3	3
B4	4
B5	5
B6	6
B7	7
B8	8
B9	9
BA	:
BB	;
BC	less than
BD	=
BE	greater than
BF	?
C0	@
C1	A
C2	B
C3	C
C4	D
C5	E
C6	F
C7	G
C8	H
C9	I
CA	J
CB	K
CC	L
CD	M
CE	N
CF	O
D0	P
D1	Q
D2	R
D3	S
D4	T
D5	U
D6	V
D7	W
D8	X
D9	Y
DA	Z
DB	left bracket
DC	black slash
DD	right bracket
DE	up arrow
DF	left arrow

SPECIALS

HEX	CONTROL
81	beginning of tape
87	bell
8A	line feed
8D	carriage return

APPENDIX B – MAC 16 MACHINE OPERATIONS

MNEMONIC	FUNCTION	LISTING TYPE	ASSEMBLY CLASS	HEXADECIMAL CODE	OPERATING TIME IN μs
ABA	Absolute Value of A	1	0	014X	2
ADC	Add Carry to A	1	0	01CX	2
ADD	Add Memory to A	2	1	8000	2
ADI	Add Immediate to A	5	5	08MN	2
ALI	Arithmetic Left Shift & Insert	4	4	0C5N	2-5
ALS	Arithmetic Left Shift	4	4	0C4N	2-5
ANA	AND Memory with A	2	1	B000	2
ARI	Arithmetic Right Shift & Insert	4	4	0CDN	2-5
ARS	Arithmetic Right Shift	4	4	0CCN	2-5
CAA	Compare A with Memory Arithmetically	2	1	F000	3
CLA	Clear A	1	0	054X	1
DNX	Decrement Index and Skip on Zero	4	4	024N	3
EAI	External Address In	0	1	06CX	3
ECO	External Command Out	3	2	0FMN	3
EDI	External Data In	3	2	0AMN	3
EDO	External Data Out	3	2	0BMN	3
ESI	External Status In	3	2	0EMN	3
HLT	Halt	1	0	00XX	1
INC	Increment Memory and Skip on Zero	2	1	E000	3
INX	Increment Index and Skip on Zero	4	4	020N	3
JIX	Jump Indirect thru Index	4	4	03CN	4
JMA	Jump Direct thru A	1	0	0470	2
JMM	Jump and Mark	2	1	4000	3
JMP	Jump	2	1	5000	2
JMX	Jump Direct thru Index	4	4	07CN	3
JRL	Jump and Reset Level	2	1	1000	2
LAX	Load A Direct from Memory Location Specified by Index	4	4	070N	3
LDA	Load A from Memory	2	1	D000	2
LDI	Load A Immediate	5	5	0DMN	1
LDX	Load Index from Memory	2	1	C000	3

MNEMONIC	FUNCTION	LISTING TYPE	ASSEMBLY CLASS	HEXADECIMAL CODE	OPERATING TIME μs
LIX	Load A Indirect thru Memory Location Specified by Index	4	4	030N	4
LLC	Logical Left Shift, Closed	4	4	03CN	2-5
LLI	Logical Left Shift & Insert	4	4	0C2N	2-5
LLN	Logical Left Shift, No Change to C	4	4	0C0N	2-5
LLO	Logical Left Shift, Open	4	4	0C1N	2-5
LRC	Logical Right Shift, Closed	4	4	0CBN	2-5
LRI	Logical Right Shift & Insert	4	4	0CAN	2-5
LRN	Logical Right Shift, No Change to C	4	4	0C8N	2-5
LRO	Logical Right Shift, Open	4	4	0C9N	2-5
LSB	Load A_{0-5} with status of C,V,R,S,B, and H	0	1	056X	1
NOP	No Operation	1	0	0480	2
ONA	One's Complement of A	1	0	018X	2
ORA	OR from Memory to A	2	1	A000	2
REX	Reset Sense Indicator X	0	1	05CN-05FN	1
SAG	Skip if A is Greater than Zero	4	3	04DN	2
SAN	Skip if A is Negative	4	3	04BN	2
SAX	Store A in Location Specified by Index	4	4	074N	3
SAZ	Skip if A is Zero	4	3	04AN	2
SBI	Subtract Immediate from A	5	5	09MN	2
SEX	Set Sense Indicator X	1	0	050N-053N	1
SIX	Store A Indirect thru Location Specified by Index	4	4	034N	4
SKN	Skip if A is Normalized	4	3	049N	2
SKP	Skip Unconditional	4	3	048N	2
SKX	Skip if Index is Zero	4	3	02CN	3
SLZ	Skip if Least-Significant Bit of A is Zero	4	3	04CN	2
SNB	Skip if Base Page Control is Zero	4	3	044N	2
SNC	Skip if C is Zero	4	3	040N	2
SNH	Skip on Interrupt not Inhibited	4	3	045N	2
SNV	Skip if V is Zero	4	3	041N	2
SNR	Skip if R is Zero	4	3	042N	2
SNS	Skip if S is Zero	4	3	043N	2
SSB	Store A_{0-5} into status bits C,V,R,S,B, and H	0	1	057X	1
STA	Store A in Memory	2	1	6000	2
STL	Store Left Byte of A in Memory	2	1	3000	2
STR	Store Right Byte of A in Memory	2	1	7000	2
STX	Store Index in Memory	2	1	2000	3
SUB	Subtract Memory from A	2	1	9000	2
TAL	Transfer A to L Register	4	4	058N	1
TLA	Transfer Level to A	1	0	046X	2
TSA	Transfer Switches to A	1	0	05AX	1
TWA	Two's Complement of A	1	0	010X	2
XXA	Exchange Index and A	1	0	060X	3

APPENDIX C – LEAP PSEUDO-OPERATIONS

PSEUDO-OPERATION (LISTING TYPE)

BOOT	BOOTSTRAP FORMAT (6)
CLEAR	CLEAR AND RESERVE IN AREA (1)
COMN	RESERVE AN AREA IN COMMON (1)
DC	DATA CONSTANT (1)
DS	RESERVE DATA STORAGE AREA (1)
DUP	DUPLICATE THE NEXT SOURCE LINE (6)
EJECT	SKIP TO THE TOP OF THE NEXT PAGE (NOT LISTED)*
END	END OF SOURCE PROGRAM (1)
ENTRY	EXTERNAL SYMBOL DEFINITION (1)
EQU	SYMBOL EQUALS VALUE AND TYPE OF EXPRESSION (7)
EQUR	SYMBOL EQUALS VALUE OF EXPRESSION, RELOCATABLE (7)
EXTRN	EXTERNAL SYMBOL (6)
LIST	LIST DURING PASS 2 (6)
LSTSY	LIST SYMBOL TABLE (6)
NLIST	DON'T LIST DURING PASS 2 (6)
ORG	SET PROGRAM ORIGIN (7)
PRINT	PRINT DURING PASS 1 (6)
PTR	ADDRESS POINTER (1)
REDEF	REDEFINE SYMBOL (1)
RESB	RESET LOADER B FLIP-FLOP (6)
SETB	SET LOADER B FLIP-FLOP (6)
SKIPF	SKIP IF FALSE (7)**
SKIPT	SKIP IF TRUE (7)**
SPACE	SPACE N LINES BEFORE RESUMING LISTING (NOT LISTED)*
TXT	CHARACTER STRING (1)
TITLE	SUB-HEADING TO BE PRINTED AT TOP OF PAGE (NOT LISTED)*

*Not listed when LIST pseudo-operation is not stated.

**When LIST pseudo-operation is not stated and condition causes coding to be skipped, neither the SKIPT (SKIPF) nor the skipped coding are listed. To list, use the LIST pseudo-operation.

LEAP Assembler Manual

Lockheed Electronics Company
Data Products Division
6201 East Randolph Street
Los Angeles, California, U.S.A. 90022
(213) 722-6810 TWX 910-580-3623
A division of Lockheed Aircraft Corporation