ASSEM is now the "standard" assembler for the CDC 3300 under OS-3. COMPASS is still available, but is no longer "supported". ASSEM is about twice as fast as COMPASS and takes much less scratch file space. It is similar to COMPASS, but there are some incompatible differences. ASSEM also has a number of features that COMPASS does not have. We shall first list the differences, then give a brief description of ASSEM's features.

Differences between ASSEM and COMPASS.

1. ASSEM is called by a control statement of the form:

```
ASSEM,D,I,L,P,R,S,X
```

The parameters have the same meanings as they do for COMPASS. There is no M parameter. The input unit is not rewound unless it is a file name, or the lun 50. If I=50 is used, ASSEM does a "set destructive read" on the unit. (This rewinds the unit, and if it is a non-saved unprotected file, puts it into destructive read mode.) Also on I=50, ASSEM unequips lun 50 when it is done.

One can use the "/R" notation on units. For example: I=20/R,L=15/R. Another difference is that ASSEM does not write file marks on its output units when done. In a batch job, if there is no D or L parameter, ASSEM will list diagnostics anyway. On teletypes and TV's, diagnostics will not be listed unless the D parameter is used. Of course, if a listing is obtained, the diagnostics are printed on it.

2. The following COMPASS pseudo-ops are not recognized by ASSEM, and will cause an O diagnostic if encountered:

```
ASCII     IFF     IFT
ENDM      IFN     IFZ
FINIS     IFP     LIBM
```

3. ASSEM does not recognize literals. For example, the following is not allowed:

```
LDA    =D1234567
```

4. ASSEM has a more general and more flexible macro capability than COMPASS, but it is not compatible. See the section on macros.

5. ASSEM is less restrictive than COMPASS on the format of source programs. In ASSEM, labels must start in column 1, but the operation code can start in any column after column 1. At least one space is required between the label (if any) and the op code field. At least one space is required between the op code field and the address field. If there is a comment, the address field must be present and at least two spaces are required between the address field and the comment. (Exception: the address field is not required on instructions that do not have address fields, such as AQE, CTI, etc.) Within the address field, single spaces are used to separate operands and operators, when the operators are words such as AND, EQ, etc.

5. (con.)

ASSEM ignores blank lines. (COMPASS assembles a zero word and flags an error on a blank line). If the input to ASSEM consists of BCD records, columns 73 to 80 are ignored, except that line numbers in columns 76 to 80 are picked up (if present) and moved over to the left side of the listing. If the input to ASSEM consists of a COSY or EDIT deck, columns 73 to 80 are included in the processing of instructions.

6. ASSEM does not allow binary or decimal scaling factors on fixed-point constants in OCT, DEC, and DECD pseudo-ops. Such constants must be written as expressions. For example:

```
          COMPASS                ASSEM
           OCT  23B12             OCT  23*2↑12D
           DEC  495B14D-2         DEC  495*2↑14 DIV (10↑2)
```

7. The power of ten scale factor on floating point constants in DECD, which is denoted by D in COMPASS, can be specified by either D or E in ASSEM. However, ASSEM does not allow a floating point constant to begin with a point. For example, DECD .025 must be written as DECD 0.025 .

8. If the IDENT pseudo-op is omitted, ASSEM does not complain, but uses blanks for the program name.

9. COMPASS allows pseudo-ops of the form ***, $$$, etc., and prints a line of stars, dollar signs, etc. (These are used to "dress up" the listing). ASSEM does not recognize these pseudo-ops and flags them as errors.

10. The principal features of ASSEM that distinguish it from COMPASS are: (1) the macro facility; (2) address expressions; (3) string variables; and (4) conditional assembly. These features are described in subsequent sections. Here we present a list of operation codes for which there are differences between ASSEM and COMPASS.

ASCII

This pseudo-op is not recognized by ASSEM. However, ASCII constants can be used in address expressions (see the section on expressions).

BCD

ASSEM allows the use of an expression to specify the number of words. For example:

```
          MESSIZE  EQU   5
          MESSAGE  BCD   MESSIZE,THIS IS AN EXAMPLE
```

BCD,C

ASSEM allows the use of an expression to specify the number of characters.

BOX

In COMPASS, BOX and EBOX produce a "box" on the listing consisting of the first character found in the address field of BOX. (If no address field is present, the box is made of asterisks). On the lines between BOX and EBOX, COMPASS replaces column 1 by blank, and puts the sides of the box in columns 2 and 72.

ASSEM ignores the address field of BOX, and always uses asterisks for the box. The sides of the box are in columns 0 and 73, so that all information from column 1 through 72 is printed.

COMMON
DATA

ASSEM can assemble information to be placed in named data blocks. To specify such blocks, use either COMMON or DATA, and put the data block name in the label field (column 1). If there is no label on COMMON or DATA, ASSEM behaves the same as COMPASS.

DEC

COMPASS allows only decimal constants in the "address" field of DEC, but it allows binary and decimal scaling factors.

ASSEM allows expressions in a DEC and each expression defines one word. Scaling factors are not allowed. Integer constants are assumed to be decimal integers unless followed by a B, in which case they are octal integers.

DECD

COMPASS allows double-word fixed point constants, with binary or decimal scaling factors, if desired. It also allows floating point constants, with a D to specify the decimal scaling factor.

ASSEM allows expressions for double-word fixed point constants, and does not allow scaling factors. Integer constants are decimal unless followed by a B, in which case they are octal. ASSEM also allows floating point constants (no expressions), with a D or E to specify the decimal scaling factor.

END

ASSEM uses END for two purposes: (1) to end macro prototypes; (2) to end a subprogram.

ENDM

ASSEM does not recognize ENDM. Use END to end macro prototypes.

ENTRY

On the listing, ASSEM prints the value of the last symbol appearing in the ENTRY declaration. This compensates for the fact that ASSEM does not print a list of entry points, as COMPASS does.

EXIT

Within a macro prototype, EXIT can be used to terminate expansion of the macro. (COMPASS does not recognize EXIT).

FINIS

ASSEM does not recognize FINIS and flags it as an error. For ASSEM, one can use a file mark, or simply end of data, to terminate the source deck.

GOTO

This pseudo-op causes ASSEM to "go to" the line with the specified label and continue assembly from that point. See the section on conditional assembly.

IDENT

For ASSEM, the IDENT pseudo-op can occur anywhere before the END card. It simply defines the subprogram name. If no IDENT card is found, ASSEM uses blanks for the subprogram name.

IF

The IF pseudo-op handles conditional assembly in ASSEM. See the section on conditional assembly.

IFF
IFN
IFP
IFT
IFZ

These COMPASS conditional pseudo-ops are not recognized by ASSEM.

INCLUDE

This ASSEM pseudo-op causes information in a specified file to be included in the program. The information can be anything that is acceptable to ASSEM, such as macro definitions, COMMON or DATA declarations, etc. The "address" field of INCLUDE must be the file name or logical unit number of the file to be included. For example, INCLUDE *IF will include the text in the public file *IF (which contains macro definitions that simulate the COMPASS pseudo-ops IFN, IFP, and IFZ).

JUMP

COMPASS does not recognize the simulated JUMP instruction, which enables transfer of control to any location in lower or upper memory. In COMPASS, one must use the octal op code 77, or use a VFD.

ASSEM recognizes JUMP, and allows a 16-bit address field.

KLUDGE

ASSEM has a string substitution feature (see the section on string variables), in which the characters $ and : are control characters. Normally this feature is off, except during macro expansions, or in case one has used the SET or RESET pseudo-ops. To enable this feature, use the pseudo-instruction KLUDGE ON. To disable it again, use KLUDGE OFF.

LIBM

ASSEM does not recognize the LIBM pseudo-op, and does not have the "macro library" feature of COMPASS. One can use INCLUDE (see description) to accomplish a similar effect.

LIST

The LIST pseudo-op, with no address field, has the same effect in ASSEM as in COMPASS (turn listing on). In ASSEM, there are two special forms of this pseudo-op. LIST DETAIL causes ASSEM to list all the words generated by DEC, DECD, OCT, etc. (It normally lists only the first word). LIST MACROS causes ASSEM to list macro expansions (which are normally not listed). Of course, no listing at all occurs unless the L option is used in the parameter string when ASSEM is called. Also see NOLIST.

LOCAL

This pseudo-op is used in ASSEM macro prototypes to declare local symbols. See the section on macros.

MACRO

Both COMPASS and ASSEM have macro capabilities, but they are very different. See the section on ASSEM's macro facility.

NAME

This pseudo-op is used in ASSEM macro prototypes to specify a name for the macro. See the section on macros.

NOLIST

This is similar to LIST, but turns off the listing. In ASSEM, one can use NOLIST to suppress listing completely, NOLIST DETAIL to turn off the "detail" listing, or NOLIST MACROS to suppress listing of macro expansions. Also see LIST.

OCT

COMPASS allows only octal constants in the "address" field of OCT, but it allows an optional binary scaling factor.

ASSEM allows expressions in an OCT and each expression defines one word. Scaling factors are not allowed. Integer constants are assumed to be octal integers unless followed by a D, in which case they are decimal integers.

PCHANGE

This ASSEM pseudo-op can be used to change the names of pseudo-ops. For example, PCHANGE EXT,EXTRN changes the EXT pseudo-op to EXTRN. This makes it possible to have a macro with the name EXT.

PRG

In ASSEM, if a label is used on PRG, subsequent lines will be assembled into a data block whose name is the label. In other words, a labeled PRG is the same as a labeled COMMON or DATA (see description).

PUNCH

This ASSEM pseudo-op causes a BCD card to be punched (written) on the same unit as the object deck (P or X parameter). The information to be punched is specified by a string, enclosed in any non-blank character. For example, PUNCH 'EXS,ERROR=2' punches a BCD card with EXS,ERROR=2 on it.

REEQU

This ASSEM pseudo-op is the same as EQU, except that it allows a symbol to be re-defined.

RESET

This ASSEM pseudo-op is the same as SET (see which), except that it allows a symbol to be re-defined.

SCAQ

ASSEM does not permit a relocatable address field on the SCAQ instruction. This limitation can be circumvented by using a VFD.

SET

This ASSEM pseudo-op defines a symbol to have a string value. It also turns the "kludge" feature on.See the section on string variables.

SHA
SHAQ
SHQ

ASSEM does not permit relocatable addresses on these shift instructions. VFD's can be used to get around this restriction.

STOP

This ASSEM pseudo-op is the same as SBJP, except that STOP allows an address field of up to 12 bits. Thus, STOP 0 is exactly the same as SBJP, while STOP 1 does a SBJP and zeroes lower memory.

VFD

ASSEM does not recognize the I (ASCII) field in VFD's. One can use ASCII constants in A or O fields, however. ASSEM allows expressions in both A (address expression) and O (octal) fields.

VFD,C

ASSEM allows a C modifier on VFD. The effect is similar to BCD,C. A VFD,C on one line can leave a word partly filled (up to a character boundary), and a VFD,C on the next line can fill that word and go on to the next one, if desired.

VFD,B

ASSEM also allows a B modifier on VFD. A VFD,B on one line can leave a word partly filled (to any bit position in the word), and a VFD,B on the next line can put more bits into that word.

Address expressions.

In COMPASS, the only operators allowed in address expressions are + and -. ASSEM handles many arithmetic and logical operations, and recognizes several forms of constants. It also allows expressions to be used in places where COMPASS permits only constants, such as the "address" fields of DEC, DECD, OCT, and O (octal) fields in VFD's. ASSEM allows any form of constant in the operation code field, provided that its value is in the range 0 to 77_8.

Expressions in ASSEM consist of operators, operands, and parentheses (), which can be nested. Operators are listed below, in order of precedence (highest to lowest).

↑

* / BYTE DIV MOD

+ -

EQ GE GT LE LT NE

AND

OR XOR

Operators that are denoted by words (such as AND, EQ, etc.) must be separated from their operands by a single space on each side. (These words can also be used as symbols in a program). Two spaces in a row terminates an address field. Operands have integer values, which may be relocatable, and operators perform integer arithmetic on these values. Relocatable values can only be added or subtracted. Absolute values can participate in any operations. Intermediate results during evaluation of an expression can be up to 3 words (72 bits) in length. A description of the various operators follows.

+ -

When used as binary operators (two operands), + denotes addition and - denotes subtraction.

When used as unary operators (one operand), + is ignored, and the - denotes change of sign (complement all bits).

*

The * can be used both as an operator, denoting multiplication, and as an operand, denoting the location of the current instruction. It is also used in the special form **.

/

The / denotes division. The division must go exactly, with zero remainder, or an A error will be flagged. Also see DIV.

↑

The ↑ denotes exponentiation (raise to power). The exponent must be an integer in the range 0 to 47.

AND

This is a logical operator, which performs a bit-for-bit logical "and" on its operands.

BYTE

This operation "forces" a number to be divisible by a specified divisor. The definition is: A BYTE B is equal to $(A \div B)*B$. (Divide A by B, discarding remainder, then multiply quotient by B).

DIV

This operation denotes division, with the remainder (if any) discarded. The only difference between DIV and / is that DIV does not "complain" about a non-zero remainder.

EQ GE GT LE LT NE

These operations compare their operands, to see if they are $=$, $\geq$, $>$, $\leq$, $<$, or $\neq$, respectively, and produce a result of 1 if the comparison is satisfied, 0 if not.

MOD

The result of this operation is the remainder from division. For example, 17 MOD 5 produces the result 2.

OR

This is a logical operation, that performs a bit-for-bit inclusive "or" on its operands.

XOR

This is a logical operation, that performs a bit-for-bit exclusive "or" on its operands.

The following kinds of items can be used as operands in an expression:

constants
symbols
*
functions

Constants can be numerical, Hollerith, ASCII, or hexadecimal. A numerical constant consists of one or more digits. It is an octal constant if it is followed by a B; it is a decimal constant if followed by a D. If neither a B nor D follows it, it is assumed to be decimal unless it appears in an OCT instruction, in an O field of a VFD, or in an operation code field. An octal constant must not contain the digits 8 or 9.

Hollerith, ASCII, and hexadecimal constants can be denoted in either of two ways. One way is to write a decimal integer specifying the number of characters, then the mode specifier (H, K, or X), and then the characters themselves. The other way is to write the mode specifier first (H, K, or X), followed by a string of characters enclosed in apostrophes ('). Any characters can appear in Hollerith or ASCII constants, except that in the second form, an apostrophe within the string must be denoted by two apostrophes in a row. Only hexadecimal characters (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) can appear in a hexadecimal constant. In all cases, the value of the constant is an integer, right-adjusted, zero filled on the left. The following table shows the number of bits per character, the maximum number of characters allowed, and the codes used for characters in each kind of constant.

Mode	Bits/char	Max no. of chars	Code
H (Hollerith)	6	8	BCD
K (ASCII)	8	6	ASCII (bit 7 = 1)
X (hexadecimal)	4	12	hexadecimal

The following examples show the octal value that results from each of several cases.

3HA'B	=	H'A''B'	=	00211422
3KA'B	=	K'A''B'	=	60323702
3X9A2	=	X'9A2'	=	00004642

Symbols have the same form as in COMPASS (one to eight letters, digits, or periods, starting with a letter). They are defined by appearing in the label field of a machine instruction or of certain pseudo-ops. In ASSEM, a symbol may also be defined by appearing in a SET or RESET instruction, which assigns a character string as a value. Symbols whose values are numeric (not string) can be used in expressions. They may be defined on a subsequent line, except in the case of certain pseudo instructions (such as BSS, EQU, etc.).

The * can be used as an operand to denote the location of the current instruction. It thus has a relocatable value.

There are three functions which can be used in expressions. In each case, the function consists of a letter (L, N, or T) denoting which function is desired, then an apostrophe(''), and then a symbol which is the argument of the function. The functions are described below.

L'symbol
The symbol must have a string value, and the value of this function is the Length (number of characters in the string).

N'symbol
The symbol must have a string value, and the value of the function is the Number of "items" in the string (number of commas plus 1).

T'symbol
This function can be used on any symbol, and its value is 0 if the symbol is undefined, 1 if the symbol has a numeric value (or has been declared external), and 2 if the symbol has a string value.

The special notation ** can be used instead of an expression in address fields that reference memory (either 15-bit word addresses or 17-bit character addresses) or in fields that refer to the register file (6-bit field). It can also be used in A, C, and O fields of a VFD. In each case, the field is filled with 1-bits. Unlike COMPASS, ASSEM does not allow the ** notation to be used in 12-bit mask fields (INS, SSIM, etc.), nor in index or channel fields.

ASSEM's expression evaluator has some peculiar quirks. If it refuses to accept an expression that it ought to, one can usually rearrange the expression (change the order of factors, for example) to make it acceptable.

Conditional assembly.

In COMPASS, one can use the pseudo-instructions IFF, IFN, IFP, IFT, and IFZ to assemble a group of instructions only if some condition is satisfied. If the condition is not satisfied, a specified number of lines is skipped (not processed).

ASSEM does not recognize any of the five pseudo-ops mentioned above. Instead, it has two pseudo-ops, IF and GOTO, which provide a more powerful conditional assembly facility than that of COMPASS. These operations are discussed below.

IF expression,line

The expression is evaluated (in pass 1). If the value is 0 (false), the "line" following the comma is ignored. If the value is non-zero (true), the information following the comma is processed just like any line of code. If the character following the comma is non-blank, then the "line" has a label. If it is blank, the next non-blank character is the operation code. The "line" to be conditionally processed can be any machine or pseudo-operation, including another IF. The expression following IF usually includes compare operators (EQ, LT, etc.), but this is not required. Here are some examples:

```
        IF A EQ B, OCT 473
        IF T'X EQ 0,X EQU *
        IF CTR GT 0, GOTO .LOOP
```

GOTO .label

This pseudo-op causes the assembler to "go to" the line on which the .label appears, process that line, and go on from there. GOTO can jump either forward or backward. The labels used with GOTO must start with a period. On the line which is labeled, the label must start in column 1 (the period must be in col. 1). The rest of the line can be an instruction, or can be left blank. Here is an example:

```
        IF A LT B, GOTO .AB
        ENA B
        GOTO .BA
   .AB
        ENA A
   .BA
```

If the value of the symbol A is less than the value of B, an ENA A is assembled; otherwise, an ENA B is assembled. (Both A and B must have absolute numerical values; relocatable values cannot be compared.)

The public file *IF contains an ASSEM macro that simulates COMPASS pseudo-ops IFN, IFP, and IFZ. It also includes an IFM (if minus). If one has a COMPASS program in which these pseudo-ops are used, one can insert an INCLUDE *IF to enable ASSEM to handle the program. The IFN, IFP, IFZ pseudo-ops become macro calls.

String variables.

In COMPASS, the formal parameters in a macro prototype are "string variables". When a macro is called, the actual parameters, which are strings of characters, are assigned to the corresponding formal parameters. As the body of the macro is processed, each reference to a formal parameter is replaced by the string which it represents. It is also possible to compare strings in the macro body, using IFF and IFT, and parts of the macro body can be assembled or skipped according to the results of the comparisons.

ASSEM also has a string-variable feature which is invoked automatically by a macro call. In ASSEM, however, this feature can be used independently of macros. One can substitute an entire string, or portions of the string (which cannot be done in COMPASS). ASSEM does not have a string comparison capability, but one can convert strings (that are not too long) to integer values and compare the integers.

In ASSEM, a macro call assigns the strings constituting the actual parameters to the corresponding formal parameters (see next section). The SET and RESET operations explicitly carry out a string assignment. The forms are:

```
          SET  symbol,string
          RESET  symbol,string
```

RESET is the same as SET except that it allows a symbol to be re-defined. The string is not enclosed in brackets. It consists of all the characters after the comma, to and including the last non-blank character in the line. String substitution is not automatic in ASSEM (as it is in COMPASS macros). The $ is used to specify that substitution is to take place. One can also use "subscripts" to specify that only a part of the string is to be substituted. The forms and actions are described below.

$symbol

If the symbol has a string value, the $ and the symbol are replaced by the entire string which is the value of the symbol. If the symbol has a numeric value, the form ($symbol) is replaced by a 4-character decimal integer representing its value.

$symbol(expression)
The symbol must have a string value. The expression is evaluated, and must produce a positive integer value. The entire form (shown above) is replaced by the n'th item in the string, where n is the value of the expression. Items in a string are separated by commas. For example, the third item consists of all characters between the second and third commas (not including the commas).

$symbol(expression,expression)
The symbol must have a string value. The expressions are evaluated, and must produce positive integer values (the second expression may be zero). The entire form (shown above) is replaced by a substring consisting of the n characters beginning with the k'th character of the string, where k (starting character position) is the value of the first expression, and n (number of characters) is the value of the second expression. In this substitution, commas are treated like any other characters.

Here are some examples.

```
        SET  S,ABC,D,*/Z
```

This assigns to S the string ABC,D,*/Z. The form $S would be replaced by the entire string. $S(2) would be replaced by D. And, $S(3,5) would be replaced by C,D,*. Note that the L and N functions can be very useful in working with strings. For example, given the value of S shown above, SET SS,$S(3) assigns to SS the string */Z. Then $SS(1,L'SS-1) would be replaced by */.

The colon (:) also plays a special role. It serves as a "vanishing delimiter". For example, SET T,ABC. Then the form $T:K5 becomes ABCK5. The colon serves to separate T and K, but disappears after the substitution is made. If one actually needs a $ or : at some point, one uses $$ or :: to represent it.

The $ and : are normally treated like any other characters. They only assume the special meanings described above when the "kludge" feature is on. This feature is turned on when a macro is called, and turned off when the macro expansion is complete. It is also turned on by SET or RESET, or by using the KLUDGE ON pseudo-op. In these cases, it remains on until a KLUDGE OFF, or until the END of the subprogram.

To compare strings, one can use the Hollerith constant feature. For example, if X has a string value, then H'$X' EQ H'ABC' would be true (1) if the value of X is ABC, and false (0) if not. However, note that H'0ABC' and H'ABC' are equal (both have the octal value 00212223). Also, no more than eight characters are allowed in a Hollerith constant.

Macros.

The conditional assembly and string variable features of ASSEM are especially useful in macros. These features make ASSEM's macro facility more powerful than that of COMPASS. In both assemblers, a macro is defined by a "prototype". In COMPASS, all macro prototypes must appear at the beginning of the subprogram. In ASSEM, a macro prototype must appear before the first call on the macro. In both assemblers, there may be some formal parameters associated with the macro. A macro call specifies actual parameters that are strings of characters, which are assigned to the formal parameters. Then the body of the macro is processed as if it were source code just being read. When a formal parameter is encountered in the macro body, the corresponding actual parameter may be substituted for it. In COMPASS, this substitution takes place automatically. In ASSEM, the substitution occurs only if a $ precedes the formal parameter, and one can use subscripts to specify that only part of the actual parameter is to be substituted (see the previous section).

The basic form of a macro prototype in ASSEM is:

```
MACRO    P1,P2,P3
NAME     symbol
...
END
```

The MACRO pseudo-op begins the prototype and specifies up to 3 formal parameters. Any valid symbols can be used as formal parameters, and any or all of them may be omitted. For example, MACRO ,,A has only one parameter, the third one, with the first two being omitted.

The NAME pseudo-op specifies a name for the macro, and also indicates where ASSEM is to begin expansion of the body. A single macro prototype can have several names. The NAME line has one of two forms:

```
NAME    symbol
NAME    symbol,line
```

The first form simply specifies a name and starting point. The second form also specifies a "line" which is the first thing processed when the macro is called by the specified name.

The END pseudo-op terminates the prototype and also ends the processing of a macro call. One can use the EXIT pseudo-op within the macro body to terminate processing.

The LOCAL pseudo-op can be used in macro bodies to declare symbols that are "local" to the macro. It has the form:

```
LOCAL    symbol,symbol, ...
```

The LOCAL pseudo-op should be executed before the first occurrence of any of the symbols declared to be local. It must follow the

NAME line, so that it will be executed during the processing of the macro body.

There is a special form of the SET and RESET pseudo-ops which can be useful in macros. The form is:

```
          SET      symbol+
          RESET    symbol+
```

As usual, RESET is the same as SET, except that it permits a symbol to be re-defined. In either form, the next line from the next lower level of processing is obtained, and is assigned to the symbol. In a "level one" macro call, this will be the next line of the source program. This makes it possible for a macro to "read" lines following the line that called the macro.

A macro call in ASSEM has the form

```
  label    name,mod    address
```

The <name> must be the symbol specified in a NAME line in a macro prototype. <label>, <mod>, and <address> are all optional. If present, they are strings of characters, which are assigned to the first, second, and third formal parameters respectively (if the formal parameters exist). The comma shown above between <name> and <mod> is included as the first character of the second parameter, and can in fact be any special character, such as %, +, etc. If there is no "first" formal parameter, but there is a label on the macro call, the label is defined as the current location (equivalent to label EQU *).

Here are some examples of macros:

Example 1.

```
          MACRO     ,,Z
          NAME      ADD, RESET OP,FAD
          NAME      SUB, RESET OP,FSB
          LDAQ      $Z(1)
          $OP       $Z(2)
          STAQ      $Z(3)
          END
```

This prototype has two names, ADD and SUB. A call such as ADD A,B,C means A+B→C, in floating point arithmetic. The expansions of two calls are shown below.

```
Call:          ADD      A,B,C

Expansion:     LDAQ     A
               FAD      B
               STAQ     C

Call:          SUB      X+2,Y,ZILCH

Expansion:     LDAQ     X+2
               FSB      Y
               STAQ     ZILCH
```

Example 2.

```
               MACRO    ,M,LUN
               NAME     BKSP,CODE REEQU 7
               NAME     CLEAR,CODE REEQU 1
               NAME     FWSP,CODE REEQU 8
               NAME     RELEASE,CODE REEQU 3
               NAME     REWIND,CODE REEQU 4
               NAME     SEFB,CODE REEQU 6
               NAME     SEFF,CODE REEQU 5
               NAME     STATUS,CODE REEQU 0
               NAME     WFM,CODE REEQU 2
               ENQ      $CODE
               CNTL$M   $LUN
               END
```

This prototype has 9 names. Calls on this macre perform various file eperations, such as REWIND, WFM, etc.

```
Call:          WFM      LUN

Expansion:     ENQ      0002
               CNTL     LUN
```

(Note: LUN in the macro prototype is local. LUN in the expansion is global).

```
Call:          REWIND,I   OUTUNIT

Expansion:     ENQ      0004
               CNTL,I   OUTUNIT

Call:          FWSP     0,2

Expansion:     ENQ      0008
               CNTL     0,2
```

Note that CODE is not local to the macro. If the line ENQ $CODE in the prototype were written ENQ CODE, then all the expansions would contain ENQ CODE and the last value assigned to CODE would be used in all cases.

Example 3.

```
               MACRO    ,M,A
               NAME     BINOCT
               LOCAL    LOOP
               ENI      $A(2)-1,1
       LOOP    ENQ      0
               SHAQ     -3
               SHQ      3
               SQCH     $A(1),1
               IF       H'$M' EQ H',SZ', ASE,S 0
               IJD      LOOP,1
               END
```

Example 3 (con).

A call such as BINOCT ADDR,N converts the number in the A-register to octal form for printing, storing N octal digits with the left-most digit at character address ADDR. If the modifier SZ is used on BINOCT, leading zerees are suppressed.

```
Call:           BINOCT    LINE+7,4

Expansion:      ENI       4-1,1
     LOOP       ENQ       0
                SHAQ      -3
                SHQ       3
                SQCH      LINE+7,1
                IJD       LOOP,1
```

(Note: no ASE,S 0 in this expansion. Leading zerees are stored.)

```
Call:           BINOCT,SZ   ZIP,7

Expansion:      ENI       7-1,1
     LOOP       ENQ       0
                SHAQ      -3
                SHQ       3
                SQCH      ZIP,1
                ASE,S     0
                IJD       LOOP,1
```

This expansion includes the ASE,S 0 to exit from the loop when (A) becomes 0, thereby suppressing leading zerees. Note that LOOP is declared local, so that the macro can be called several times, and the references to LOOP in each call do not conflict with those in other calls.

Example 4.

```
                MACRO     ,,LUNS
                NAME      UNEQUIP
                LOCAL     CTR
     CTR        EQU       1
     .LOOP
                ENI       2,1
                XREQ      $LUNS(CTR)
     CTR        REEQU     CTR+1
                IF        CTR LE N'LUNS, GOTO .LOOP
                END
```

A call on this macro unequips one or several logical units.

```
Call:           UNEQUIP   X

Expansion:      ENI       2,1
                XREQ      X
```

(X is presumably equated to an integer denoting a logical unit number).

Example 4 (con.)

```
Call:          UNEQUIP   34,9,50

Expansion:     ENI       2,1
               XREQ      34
               ENI       2,1
               XREQ      9
               ENI       2,1
               XREQ      50
```

In this macro, the local symbol CTR is used as a subscript in $LUNS(CTR) to select successive items in the address field. The function N'LUNS tells how many items there are.

ASSEM does not list macro expansions unless the pseudo-op LIST MACROS has been executed. If this operation is not used, then the octal form of the first word generated by each macro call is printed at the left of the line containing the call.