THE MACRO LANGUAGE FOR THE IBM 360/370.

By: John Cox

DCS-TR-46

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, New Jersey 08903

July 1976
IN ASSEMBLY LANGUAGE WE HAVE SO FAR ENCOUNTERED TWO TYPES OF INSTRUCTIONS.

MACHINE LANGUAGE INSTRUCTIONS WHICH, WHEN PROCESSED BY THE
ASSEMBLER, PRODUCE MACHINE CODE. EXAMPLE: L 3, ALPHA A 4, BETA.

MVC 0(446)81(10). A SECOND TYPE ARE THE ASSEMBLER
INSTRUCTIONS WHICH, WHEN PROCESSED BY THE ASSEMBLER, SOMETIMES
PRODUCE CONSTANTS AND SOMETIMES JUST INFORM THE ASSEMBLER
OF ACTIONS TO BE TAKEN. FOR EXAMPLE, A DC OR DS, RESERVE
STORAGE AND ASSEMBLE A CONSTANT INTO A LOCATION. WHILE
USING AND END AND TWO INFORM THE ASSEMBLER BUT DO NOT PRODUCE
ANYTHING IN THE WAY OF CODE.

WE NOW INTRODUCE A THIRD TYPE OF INSTRUCTION CALLED A
MACRO. A MACRO IS WRITTEN JUST LIKE AN ASSEMBLER INSTRUCTION,
BUT THE OUTPUT, WHEN PROCESSED BY THE ASSEMBLER IS ASSEMBLY
LANGUAGE. THE OUTPUT MAY BE MACHINE INSTRUCTIONS, ASSEMBLER
INSTRUCTIONS OR A COMBINATION OF BOTH. THIS PROCESS IS
CALLED THE EXPANSION OF THE MACRO. THE OUTPUT PRODUCED BY
THE EXPANSION OF THE MACRO IS THEN PROCESSED BY THE ASSEMBLER
JUST AS IF YOU HAD WRITTEN THE EXPANSION EXPLICITLY. THE
PURPOSE OF A MACRO IS TO SAVE YOU WRITING MANY UNNECESSARY
DETAILS THAT ARE REQUIRED BY THE SYSTEM, THUS CUTTING DOWN
ON POSSIBLE ERRORS.

IBM HAS SUPPLIED A SERIES OF MACROS FOR BOTH DATA
MANAGEMENT AND FOR SUPERVISOR SERVICES. DATA MANAGEMENT
MACROS ARE FOR INPUT/OUTPUT OF DATA. THEY INCLUDE:
GET, PUT, OPEN, CLOSE, QUIRK, READ, WRITE, OCCU, SNAP,
AND PUTX. SUPERVISOR SERVICES OF MACROS ARE FOR
CONTROL OF YOUR PROGRAM AND STORAGE. THEY INCLUDE:
SAVE, RETURN, TIME, GETMAIN, SLEEP, STAE, AND ASCEND.

THERE ARE, OF COURSE, MANY OTHERS, BUT WE WILL BE
CONCERNED WITH THESE FOR THE TIME BEING.

BEFORE GOING INTO TOO MUCH DETAIL ON MACROS, LET US
EXAMINE TWO SIMPLE MACROS, SAVE AND RETURN.

YOUR PROGRAM IS TREATED LIKE A SUBROUTINE TO THE
OPERATING SYSTEM. YOUR PROGRAM IS REACHED BY A CALL
14112 INSTRUCTION. REG 15 CONTAINS THE ENTRY POINT.
THE RETURN ADDRESS IN US, REG 0 AND 1 MAY CONTAIN
PARAMETERS OR POINT TO PARAMETER LISTS FOR YOUR
PROGRAM. IN ADDITION REG 12 CONTAINS THE ADDRESS OF
AN 16 WORD AREA OF STORAGE IN US THAT YOU ARE TO SAVE
ALL THE REGISTERS WHILE YOUR PROGRAM IS EXECUTING AND
THEN RESTORE THE ORIGINAL CONTENTS BEFORE ISSUING A
ER 14 TO RETURN TO US.

THE SAVE MACRO FACILITATES THE SAVING OF THE REGISTERS.
WHILE THE RETURN MACRO FACILITATES THE RESTORING OF THE
REGISTERS. THE RETURN MACRO HAS THE ADDITIONAL BENEFIT
OF GENERATING THE ER 14 INSTRUCTION FOR YOU.

SAVE (14412)
+DIM 14, 12+12(13)
RETURN, (14,12)
+LM  14,12,12(13)
+BR   14

The generated instructions, _LM_ and _BR_ are flagged by a + to indicate they are the result of a macro expansion. The save macro expands to the one instruction _STM_.

14,12,12(13) while the return macro expands to the two instructions _LM_. 14,12,12(13) and _BR_. 14. *Clearly it is easier to write save and return rather than the actual machine instruction generated.*

A macro is written just like an assembler instruction. COL 1 is either a symbol or a blank. COL 10 is the macro name. At least 1 blank must be on either side of the macro name. After the blank after the macro name follow zero or more operands separated by commas. Omitted positional operands must include commas to indicate the correct position. Keyword operands may be in any order. Suboperands are enclosed in parentheses and may be positional or keyword.

Most macros allow a great variety in the specifications of values for operands. Many values can be specified in several ways. In general, most of the following are valid for values of operands:

SYM 
Any symbol valid in the assembler language
ALPHA, BETA, GAMMA, DELTA(7)

RX TYPE 
Any address that can be used in an RX TYPE instruction
0(4) 813,21 0 or any of the type SYM

DEC DIG 
Any decimal digit
2 4 12 14

REGISTER 
Always quoted in parentheses
(5-12) One of general registers 2 through 12, previously loaded with the right-adjusted value or address specified in the associated macro description. The unused high order bits must be set to zero.
(1) General register 1, previously loaded as indicated above.
(0) General register 0, previously loaded as indicated above.

ADCON 
Any address that may be written in an A-TYPE address constant may be designated.

MACHUS may be continued on more than one card by stopping at any comma, placing a non-blank in col 72, and starting on the next card in col 10. These columns and rules must be strictly adhered to.

In addition to the IEM supplied macros it is possible for you to write your own macros. You write a macro definition and then write the macro call anywhere in the program. The macro
IS EXPANDED JUST AS SYSTEM MACROS ARE EXPANDED. IT IS ALSO POSSIBLE TO PASS PARAMETERS TO THE MACRO WHICH MAY ALTER THE EXPANSION ACCORDING TO THE WAY YOU WRITE THE MACRO. MACROS MAY BE DEFINED JUST ONCE IN A PROGRAM AND THEN USED SEVERAL TIMES WITH DIFFERENT PARAMETERS. A MACRO DIFFERS FROM A SUBROUTINE IN THE WAY IT IS CALLED AND THE WAY THE PARAMETERS ARE HANDLED. LET US SEE HOW A MACRO IS EXPANDED.

- SUPPOSE WE HAVE THE FOLLOWING DEFINITION FOR A MACRO:
  
  MACRO
  CALL
  LOAD
  ADD
  STORE
  MEND

  AND THE FOLLOWING INSTRUCTIONS CODED:

  INST1
  MAC
  INST2

  THE RESULT OF THE EXPANSION OF THE MACRO WOULD BE:

  INST1
  + LOAD
  + ADD
  + STORE
  INST2

  THE INSTRUCTIONS IN THE BODY OF THE MACRO ARE INSERTED AT THE PLACE WHERE THE MACRO WAS CALLED. AS WITH SYSTEM MACROS THE EXPANDED INSTRUCTIONS ARE FLAGGED WITH A + SIGN. THE RESULT OF THE EXPANSION MAY BE ANY ASSEMBLER LANGUAGE INSTRUCTIONS OR ANOTHER MACRO WHICH IS ALSO EXPANDED. Thus THE DEFINITION OF A MACRO MAY INCLUDE MACHINE INSTRUCTIONS SUCH AS ADD, SUBTRACT, LOAD OR STORE, ASSEMBLER INSTRUCTIONS SUCH AS DC, DS, OR USING ANOTHER MACRO INSTRUCTION, WHICH WOULD BE CALLED AN INNER MACRO. SUPPOSE WE HAVE ANOTHER MACRO NAMED MAC1:

- MACRO
  MAC1
  INST3
  MEND

  THEN THE EXPANSION OF:

  INST1
  MAC1
  INST2

  WOULD BE:

  INST1
THE INNER MACRO EXPANDS INSIDE THE OUTER MACRO. A MACRO DEFINITION IS WRITTEN IN THE FOLLOWING WAY: FIRST THE WORD MACRO STARTING IN COLUMN 10. NEXT THE PROTOTYPE STATEMENT WHICH IS THE WAY THE MACRO WILL BE CALLED. IF THE CALL OF THE MACRO IS TO HAVE A LABEL OR ANY PARAMETERS THEY ARE DEFINED IN THE PROTOTYPE STATEMENT. IN THE ABOVE CASES MAC AND MAC1 WERE THE PROTOTYPE STATEMENTS. NEXT COMES THE BODY OF THE MACRO. THE BODY CAN CONTAIN ANY ASSEMBLER INSTRUCTIONS AND ANY MACRO LANGUAGE INSTRUCTIONS. MACRO LANGUAGE INSTRUCTIONS WILL BE DEFINED PRECISELY LATER. BUT IN GENERAL THERE ARE INSTRUCTIONS FOR DECLARING VARIABLES, ASSIGNING VALUES TO VARIABLES, UNCONDITIONAL BRANCHING, AND CONDITIONAL BRANCHING. FINALLY THE MACRO DEFINITION IS TERMINATED WITH THE WORD MEND. THUS A MACRO DEFINITION LOOKS LIKE:

MACRO
    PROTOTYPE STATEMENT
    BODY OF MACRO
    MEND

MACRO DEFINITIONS MAY BE PLACED IN THE SOURCE PROGRAM OR IN SPECIAL LIBRARIES. IF THEY ARE DEFINED IN THE SOURCE PROGRAM, THEY MUST BE WRITTEN BEFORE ANY CSECTS. IF A MACRO IS DEFINED IN A LIBRARY THEN AN APPROPRIATE DD CARD MUST BE INCLUDED IN THE JCL TO DEFINE THE MACRO LIBRARY. WE WILL ONLY CONSIDER MACROS WRITTEN IN THE SOURCE CODE FOR NOW. OUR PROGRAMS LOOK LIKE:

MACRO
    DEFINE A MACRO
    MEND

MACRO
    DEFINE A SECOND MACRO
    MEND

CSECT
    PROGRAM INCLUDING MACRO CALLS

USEECT
    FIRST DUMMY SECTION

DSEECT
    SECOND DUMMY SECTION

END STATEMENT

DURING THE ASSEMBLY PROCESS THE SOURCE CODE IS FIRST PASSED TO A PREPROCESSOR CALLED A MACRO PROCESSOR. THE MACRO PROCESSOR EXAMINES EACH STATEMENT. IF THE STATEMENT IS NOT A MACHINE INSTRUCTION OR AN ASSEMBLER INSTRUCTION, THEN IT IS ASSUMED TO BE A MACRO. THE DEFINITION IS LOOKED UP FROM THE MACROS DEFINED IN THE SOURCE CODE. IF THE DEFINITION IS NOT FOUND THERE THEN THE MACRO LIBRARIES ARE SEARCHED. THE ORDER
THE MACRO LIBRARIES DEPENDS ON THE JCL. IF THE
DEFINITION IS FOUND THEN THE MACRO IS EXPANDED IN THE WAY
PREVIOUSLY MENTIONED. IF NO DEFINITION IS FOUND, THEN AN
ERROR IS FLAGGED. IN THIS WAY WE CAN TEST AN UPDATED VERSION
OF A MACRO BEFORE IT IS PLACED IN A MACRO LIBRARY.

TWO TYPES OF COMMENT STATEMENTS MAY APPEAR IN A
MACRO DEFINITION. IF YOU CODE A * IN COLUMN 1 THEN THE
COMMENT IS WRITTEN WHEN THE MACRO IS EXPANDED. IF YOU CODE
** IN COLUMNS 1 AND 2 THEN IT IS A COMMENT TO THE MACRO AND
NOT INCLUDED IN THE EXPANSION OF THE MACRO. ANY COMMENTS
INCLUDED ON ANY ASSEMBLER STATEMENTS ARE INCLUDED IN THE
EXPANSION OF THE MACRO.

VARIABLE SYMBOLS ARE USED BY THE MACRO PROCESSOR AS
SYMBOLIC PARAMETERS, SYSTEM SYMBOLS AND SET SYMBOLS. SEQUENCE
SYMBOLS ARE USED TO BRANCH TO WITHIN THE MACRO. VARIABLE
SYMBOLS ARE DEFINED:

A) TWO TO EIGHT CHARACTERS
B) THE FIRST CHARACTER IS AN AMPERSAND &
C) THE SECOND CHARACTER MUST BE A LETTER
D) THE REMAINING CHARACTERS MAY BE LETTERS OR DIGITS
E) THE 2-4 CHARACTERS SHOULD NOT BE SYS, SINCE THEY
DEFINE SYSTEM VARIABLES.

VALID EXAMPLES OF VARIABLE SYMBOLS ARE:
REG  GAREA  GLOC
ELOC1  ESYM  INDEX
GLABEL  GNAMES  GLIST

SEQUENCE SYMBOLS ARE DEFINED IN THE SAME MANNER EXCEPT THAT
THE FIRST CHARACTER IS A PERIOD. VALID EXAMPLES ARE:
*NOTE  **END  ***MORE
*NEAT  *AGAIN  *LOOKUP

PARAMETERS MAY BE PASSED TO MACROS. THE VALUE OF THE PARAMETER
IS USED IN THE EXPANSION OF THE MACRO. PARAMETERS ARE DEFINED IN
THE PHOTYPPE STATEMENT. THEY MAY BE POSITIONAL OR KEYWORD. ON
KEYWORD PARAMETERS DEFAULT VALUES MAY BE ASSIGNED IF THE PARAMETER
IS OMITTED.

LET US TAKE AS AN EXAMPLE A MACRO WHICH PRODUCES CODE TO
LOAD A REGISTER, ADD A NUMBER AND STORE THE RESULT. THE EXPECTED
CODE SHOULD BE:

EXI  L  5*ALPHA
A  SYLTA
ST  5*GAMMA

THE MACRO COULD BE CODED:

MACRO
ENAME ADDS  LREG*ELOC1*ELOC2*ELOC3
ENAME L  LREG*ELOC1
A  LREG*ELOC2
ST  LREG*ELOC3
WE CAN SEE THAT DIFFERENT CALLS OF THE SAME MACRO PRODUCE DIFFERENT LABELS. HAD WE NOT HAD THE FACILITY OF DEFINE THEN THE SAME LABELS WOULD BE PRODUCED WITH DIFFERENT CALLS AND THUS WE WOULD HAVE MULTIPLE DEFINED SYMBOLS IN THE PROGRAM. IF THE MACRO IS ONLY TO BE CALLED ONCE IN A PROGRAM THEN WE DO NOT NEED THIS FACILITY. USUALLY MACROS ARE DESIGNED TO BE CALLED MORE THAN ONCE IN A PROGRAM.

LET US TAKE AS AN EXAMPLE A COMBINATION OF OUR ADDS AND OUR ADDNS_MACROS. WE HAVE TWO MACROS THAT DIFFER IN THE NUMBER OF PARAMETERS. IF WE SPECIFY A REGISTER AND 3 LOCATIONS THEN WE USE THE ADDS_MACROS, BUT IF WE SPECIFY A REGISTER AND ONLY 2 LOCATIONS THEN WE HAVE TO USE THE ADDNS_MACROS. WHAT WE WOULD LIKE IS ONE MACRO NAMED ADD THAT PRODUCES THE CORRECT CODE IN EITHER CASE. IF WE CALL THE MACRO WITH 4 ARGUMENTS (A REGISTER AND 3 LOCATIONS) THEN WE WANT TO PRODUCE THE CODE FROM ADDS, BUT IF WE CALL THE MACRO WITH ONLY 3 ARGUMENTS (A REGISTER AND 2 LOCATIONS) THEN WE WANT TO PRODUCE THE CODE FROM THE ADDNS_MACRO.

WHAT IS REQUIRED IS THE ABILITY TO TEST TO DETERMINE IF THE 4TH ARGUMENT IS NULL, THAT IS OMITTED. IF IT IS NULL WE WANT TO SKIP THE GENERATION OF THE STORE INSTRUCTION. WE WILL USE THE CONDITIONAL BRANCH MACRO INSTRUCTION AIF WHICH HAS THE FORM:

```
AIF (CONDITION), SEGSYM
```

THE (CONDITION) IS EVALUATED. IF IT IS TRUE THEN A BRANCH IS TAKEN TO THE SEQUENCE SYMBOL SEGSYM AND IF THE CONDITION IS FALSE THE NEXT INSTRUCTION IS TAKEN. THIS ACTS JUST LIKE AN IF (CONDITION) GO TO INSTRUCTION IN PL/I OR FORTRAN. THE CONDITION IS WRITTEN AS:

```
(EXPRESSION RELATION EXPRESSION2)
```

THE RELATIONS ARE:

<table>
<thead>
<tr>
<th>&lt;</th>
<th>LEQUAL</th>
<th>&gt;</th>
<th>GEQUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT EQUAL</td>
<td>LESS THAN</td>
<td>LESS THAN OR EQUAL</td>
<td>GREATER THAN</td>
</tr>
<tr>
<td>GREATER THAN OR EQUAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE EXPRESSIONS MAY BE CHARACTER EXPRESSIONS OR ARITHMETIC EXPRESSIONS, BUT BOTH MUST BE OF THE SAME TYPE.

<table>
<thead>
<tr>
<th>MACRO</th>
<th>NAME</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>GREG,LOC1,LOC2,LOC3</td>
<td></td>
</tr>
<tr>
<td>ADDNS</td>
<td>GREG,LOC1</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>GREG,LOC2</td>
<td></td>
</tr>
<tr>
<td>AIF</td>
<td>((LOC3 &lt; 73)) .EQ .0) .DO</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>GREG,LOC3</td>
<td></td>
</tr>
</tbody>
</table>
The AIF statement test whether the 4th argument, BLOC3, is null. The null string is specified as the single quotes with no space between them. Since null is a character expression, we need the single quotes around "BLOC3". If the condition is true, that is "BLOC3" is null then we skip the Store Instruction and branch to the Menu. If it is false then we generate the Store Instruction and then perform the Menu. The sequence Symbol DONE is written in the label field of the Menu statement.

ADD 0,A,B,C
+ L 6,A
+ A 6,B
+ ST 0,C

ADD 7,M,N
+ L 7,M
+ A 7,N

We see that the presence or absence of the 4th argument determines whether the store instruction is generated. The AIF statement itself generates no code. It is used to determine flow of control within the expansion of the Macro.

When calling a Macro it is often convenient to treat several arguments as one sublist of arguments. A sublist consists of one or more arguments enclosed in parentheses and separated by commas. The entire sublist including the parentheses is considered to be one argument.

If B is a symbolic parameter, then B(1) refers to the first argument of the sublist while B(2) refers to the second argument of the sublist.

Consider the following Macro which adds three locations and stores the result in SUM:

```
MACH
# NAME ADD3 BLOC,REG, & SUM
# NAME L BREG & BLOC(1)
# NAME A BREG & BLOC(2)
# NAME A BREG & BLOC(3)
# NAME ST BREG & SUM
MEND
```

ADD3 (A, B, C), S, ALPHAM
+ L 5,A
+ A 5,E
+ A 5,C
+ ST 5,ALPHA

The argument that corresponds to the symbolic parameter BLOC consists of three arguments in a sublist enclosed in parentheses. BLOC(1) refers to the first argument in the sublist or in A, BLOC(2) refers to the second argument in the
MACROS

SUBLIST OR &x, while ELOC(3) refers to the third argument of the SUBLIST OR X.

We now see the reason why a period must be used when concatenating a variable symbol and a left parenthesis. ELOC(2) refers to an argument in the SUBLIST, while ELOC(2) refers to the entire list of arguments concatenated with (2). This would result in (A,B,C) (2) being generated.

MACRO
    ADDGAD   ELOC5@REG5@SUM
    L       @REG5@ELOC5(1)

    ADDGAD   (A,B,C)5@ALPHA
    L       @5(A,B,C)11

This results in an invalid expression generated for the LOAD instruction. The macro processor did its job properly, but the result is not what you want.

Another method of referring to positional symbolic parameters is by the special built-in system variable &SYSLIST. &SYSLIST(1) refers to the first positional parameter, while &SYSLIST(N) refers to the Nth positional parameter. If any parameter has AS its argument a SUBLIST then &SYSLIST(N,M) would refer to the Mth argument in the Nth SUBLIST. In the above macro there are 3 arguments, the first of which is a SUBLIST.

&SYSLIST(1)   (A*B*C)
&SYSLIST(2)   D
&SYSLIST(3)   ALPHA
&SYSLIST(1,1)   A
&SYSLIST(1,2)   B
&SYSLIST(1,3)   C
&SYSLIST(0)   THE LABEL

We may use &SYSLIST(N) in any place where we refer to a symbolic parameter. The following example illustrates this:

MACRO
&NAME   ADDS   ELOC5@REG5@SUM
&SYSLIST(0)   L   &SYSLIST(2)@&SYSLIST(1,1)
A   &SYSLIST(2)@&SYSLIST(1,2)
A   &SYSLIST(2)@&SYSLIST(1,3)
ST   &SYSLIST(2)@&SYSLIST(3)
MENU

We will see better examples of &SYSLIST later when the subscript can be a variable.

We are just beginning to see the real power of the MACRO language.
The ability to pass parameters to a macro and to concatenate the parameter to characters in the macro to form new labels, operations, and operands allows us to write macros for many of the "housekeeping" functions that we normally have to perform in a program. However, the real power of the macro language is its ability to define symbols that are local to the macro, assign values to those symbols and to make complex tests on the values of those symbols or on parameters passed to the macro. In brief, the macro language is a sublanguage to the assembly language. We will write little programs that produce the code for our assembly language programs. It takes some facility to master the macro language even though there are just a few statement types in the language.

The macro language allows the programmer to define variable symbols, assign values to these variable symbols, and to test the values of the variable symbols. A variable symbol differs from a symbolic parameter in four ways:

1) In the definition, a parameter is defined in the prototype statement. A variable symbol is defined explicitly in the body of the macro.

2) How they are assigned values. A parameter is assigned a value when the macro is called by the argument. A variable symbol is assigned a value in the body of the macro.

3) Whether the value can be changed. A parameter cannot change its value changed in the body of a macro. A symbolic variable can have its value changed in the body of a macro.

4) They can be local or global. A parameter is always local to the macro definition. The same parameter name can be in several macros. A symbolic variable can be local or global. If it is local then it is the same as a parameter. If it is global then the value assigned in one macro can be used in another.

Variable symbols can be of three types: arithmetic, binary or boolean, and character. There is a certain amount of automatic conversion that takes place that may or may not be desirable. They are defined and initialized by the following macro language statements:

LCL LOCAL ARITHMETIC INITIAL VALUE 0
LCLB LOCAL BINARY OR BOOLEAN INITIAL VALUE 0
LCLC LOCAL CHARACTER INITIAL VALUE NULL

GLAL GLOBAL ARITHMETIC INITIAL VALUE 0
GALB GLOBAL BINARY OR BOOLEAN INITIAL VALUE 0
GALC GLOBAL CHARACTER INITIAL VALUE NULL

Global variables are initialized when they are first encountered by the assembler. Subsequent definitions of global variables do not reinitialize them. Local variables are initialized each
TIME THEY ARE DEFINED, THE VALUES DO NOT CARRY FROM ONE MACRO TO ANOTHER.

LOCAL AND GLOBAL VARIABLES MAY BE REASSIGNED NEW VALUES BY THE FOLLOWING MACRO LANGUAGE STATEMENTS:

$SETA$ ASSIGN ARITHMETIC VALUE
$SETB$ ASSIGN BINARY OR BOOLEAN VALUE OF 0 OR 1, 1 FOR TRUE, 0 FOR FALSE
$SETC$ ASSIGN A CHARACTER VALUE

AIF, AGO, AND ANOP MACRO STATEMENTS MAY BE USED TO TEST VALUES AND ALTER THE GENERATION OF STATEMENTS BY THE MACRO, THEY ARE USED IN CONJUNCTION WITH SEQUENCE SYMBOLS.

WE WILL ONLY CONSIDER LOCAL VARIABLES AT THIS POINT.

THE EXACT FORM OF THE DEFINITION OF VARIABLE SYMBOLS IS:

$LCLA$ ONE OR MORE
$LCLB$ VARIABLE
$LCLC$ SYMBOLS

THE NAME FIELD MUST BE BLANK. THE VARIABLE SYMBOLS MUST NOT BEGIN WITH THE 4 CHARACTERS $CSYS$.

$LCLA$ $L1$ DEFINES A LOCAL ARITHMETIC VARIABLE
$LCLB$ $L1$ AND $L2$ DEFINES 2 LOCAL BINARY OR BOOLEAN VARIABLES
$LCLC$ $L1$, $L2$ DEFINES 2 LOCAL CHARACTER VARIABLES

ALL LOCAL DEFINITIONS MUST APPEAR IMMEDIATELY AFTER THE PROTOTYPE STATEMENT AND BEFORE ANY OTHER STATEMENTS. LOCAL VARIABLES ARE ALWAYS AUTOMATICALLY INITIALIZED WHEN DEFINED. LOCAL VARIABLES MUST BE DEFINED IN A MACRO BEFORE THEY CAN BE USED.

LOCAL VARIABLES MAY BE REASSIGNED VALUES BY $SETA$, $SETB$, OR $SETC$ STATEMENTS. THE EXACT FORM IS:

$VAR$ $SETA$ ARITHMETIC EXPRESSION
$VAR$ $SETB$ (BOOLEAN OR BINARY EXPRESSION)
$VAR$ $SETC$ CHARACTER EXPRESSION

$L1$ $SETA$ 1 ASSIGN 1 TO $L1$
$L1$ $SETA$ $L1$+1 INCREMENT VARIABLE $L1$
$L1$ $SETA$ $L1$-1 DECREMENT $L1$
$SU$ $SETA$ $SU$+4 ADD 4 TO $SU$ AND ASSIGN TO $SU$
$LEN$ $SETA$ $L$'S PARAM ASIGN TO LEN THE LENGTH ATTRIBUTE OF PARAMETER $L$'S PARAM

ARITHMETIC EXPRESSIONS ARE DEFINED IN THE MOST USUAL WAYS, THEY CAN BE A SINGLE TERM OR AN ARITHMETIC COMBINATION OF TERMS. THE ARITHMETIC OPERATORS ARE $+$, $-$, $*$, $/$: ADDITION, SUBTRACTION, MULTIPLICATION, AND DIVISION. EXPRESSIONS ARE EVALUATED LEFT TO RIGHT WITH MULTIPLICATION AND DIVISION EVALUATED BEFORE ADDITION AND SUBTRACTION.
AND SUBTRACTION, PARENTHESES MAY BE USED TO REDEFINE THE ORDER
OF EVALUATION. A MAXIMUM OF 10 TERMS MAY BE USED IN AN EXPRESSION
AND A MAXIMUM NEST-OF-5 LEVELS OF PARENTHESES MAY BE USED. THE
RESULT OF THE EVALUATION IS A SIGNED INTEGER.

THE ARITHMETIC VALUE OF A SETA SYMBOL IS USED IN AN
ARITHMETIC EXPRESSION IF THE SETA SYMBOL IS NOT USED IN AN
ARITHMETIC EXPRESSION THEN IT IS CONVERTED TO AN UNSIGNED
INTEGER WITH LEADING ZEROS REMOVED. IF THE VALUE IS ZERO
THEN A SINGLE 0 IS USED. SETA SYMBOLS MAY BE USED IN BOOLEAN
EXPRESSIONS OR CHARACTER EXPRESSIONS OR CONCATENATED IN THE
BODY OF A MACRO.

```
MACRO
ENAME  LOST              GFROMEC
LCLA  6A&6B
LCLA  6G&6D
6A  SETA  10
6F  SETA  12
6C  SETA  6A-6B
6D  SETA  6A+6C
ENAME  L  2,GFROMEC
SI  2,GLHRD
MEND

HERE  LOST              ALPHA,BETA
HERE  2,ALPHA+Z
+  SI  2,BETA+Z
```

THE VALUES ARE DETERMINED AS FOLLOWS: 6A AND 6B ARE SET TO
THE VALUES 10 AND 12 RESPECTIVELY IN THE SETA STATEMENTS. 6C
IS ASSIGNED THE VALUE OF -2 AS A RESULT OF THE ARITHMETIC EXPRESSION
6A+6C.  6D IS ASSIGNED THE VALUE TO AS A RESULT OF THE ARITHMETIC
EXPRESSION 6A+6C.  ALPHAZ IS A RESULT OF THE CONCATENATION OF
GFROMEC.  GFROMEC HAS THE VALUE ALPHAZ AND 6C IS CONVERTED TO AN
UNSIGNED INTEGER 2 WITH LEADING ZEROS STRIPPED OFF.  BETAZ IS
A RESULT OF THE CONCATENATION OF LHRD.  6T0 HAS THE VALUE BETA
AND 6D HAS THE VALUE 8 WITH THE SIGN REMOVED.

IN THE FOLLOWING EXAMPLE WE WILL GENERATE CODE TO SET A
RANGE OF REGISTERS TO 0. THE MACRO WILL GENERATE A SR RK
INSTRUCTION FOR EACH REGISTER IN THE RANGE. A WRAP-AROUND
FEATURE HAS BEEN BUILT INTO THE MACRO.

```
CLREG  14,12
+  SR  14,14
+  SR  12,12
+  SR  0,0
+  SR  1,1
+  SR  2,2

MACRO
CLREG  14,12
LCLA  LCOUNT
6COUNT  SETA  6R1
*AGAIN  ANUP
```
SR  &COUNT &COUNT
AIF  ((&COUNT EO &R2) * DONE
AIE  ((&COUNT EO &15) * RESET
&COUNT SETA &COUNT+1
AG0  *AGAIN
RESET ANDP 0
&COUNT SETA 0
AG0  *AGAIN
DONE  END

&COUNT IS A LOCAL ARITHMETIC VARIABLE. IT IS ASSIGNED THE VALUE OF THE FIRST REGISTER OF THE ARGUMENT LIST. THE ANOP WORKS LIKE AN AGAIN. ENDU

*y giving us a sequence symbol to branch to when we encounter the ag0 *again statement.
 the sr &count &count generates the instruction. a test is made to determine if we have generated the last instruction. if so, then branch to & done or the end of the macro. the next aif tests for the wrap around feature. if &count is equal to 15, we branch to &reset which again is just a place to branch to. here we reset &count to 0 and branch back to *again to generate the next instruction.

it can be seen that this macro is a little program.
with some logic built into it, the sequence of statements performed and the instructions generated are controlled by the aif and ag0 statements. the variable &count is either incremented by 1 each time through the loop or is reset to 0 when it reaches the value 15. &count is also used in the generation of the instruction sr &count &count.

recall the method of setting an area to blanks. we set the first byte to blank and then propagate the blank by an ivc with a length of 1 less than the length of the area. let is write a macro to perform that function.

MACRO

NAME CLEAR @AREA  @LEN
LCL &COUNT

NAME MV1 @AREA  @C1  CLEAR THE FIRST BYTE
.COUNT SETA &LEN-1 SET THE LENGTH MINUS 1
.MVC @AREA+1(&COUNT)*LEN AREA CLEAR THE REST OF THE AREA

MENU

CLEAR ALPHA &LEN
.MVI ALPHA &C1  CLEAR THE FIRST BYTE
.MVI ALPHA+1(79)  ALPHA CLEAR THE REST OF AREA

CHARACTER EXPRESSIONS ARE DEFINED AS UP TO 255 CHARACTERS INCLUDED IN QUOTES. LARGER STRINGS CAN BE FORMED BY CONCATENATING CHARACTER STRINGS WITH OTHER CHARACTER STRINGS, WITH SYMBOLIC PARAMETERS, OR WITH CHARACTER VARIABLES. SMALLER SUB-STRINGS MAY BE EXTRACTED FROM LARGER CHARACTER STRINGS BY SPECIFYING A
STRING, STARTING POSITION AND LENGTH. CHARACTER VARIABLES ARE
DEFINED BY THE LLCI STATEMENT. SEVERAL VARIABLES MAY BE DEFINED
WITH THE LLCI STATEMENT JUST AS IN ARITHMETIC VARIABLES. THE
VALUES OF CHARACTER VARIABLES ARE CHARACTER STRINGS. VALUES ARE
ASSIGNED BY THE SETC STATEMENT. THE MAXIMUM NUMBER OF CHARACTERS
THAT CAN BE ASSIGNED TO A SETC SYMBOL IS 8.

THE OPERAND OF A SETC STATEMENT CAN BE MADE UP OF ONE OF THE
FOLLOWING:

1) A TYPE I SYMBOL. IT WILL BE DEFINED LATER.
2) A CHARACTER EXPRESSION ENCLOSED IN QUOTES WHICH IS MADE
UP OF THE FOLLOWING:
   (a) ANY STRING OF CHARACTERS
   (b) SYMBOLIC PARAMETER OR VARIABLE SYMBOL OF ANY TYPE
   (c) CONCATENATION OF (a) AND (b)
   (d) SUBSTRING NOTATION
   (e) CONCATENATION OF 2) AND 3) ABOVE.

<table>
<thead>
<tr>
<th>LLC</th>
<th>SETC</th>
<th>EXPRESSSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>6C1</td>
<td>SETC</td>
<td>&quot;ABCD&quot;</td>
</tr>
<tr>
<td>6C2</td>
<td>SETC</td>
<td>&quot;AB&quot;, &quot;CD&quot;</td>
</tr>
<tr>
<td>6C3</td>
<td>SETC</td>
<td>&quot;6C1&quot;</td>
</tr>
<tr>
<td>6C4</td>
<td>SETC</td>
<td>&quot;6C1&quot;,&quot;6C2&quot;</td>
</tr>
<tr>
<td>6C5</td>
<td>SETC</td>
<td>&quot;6C1&quot;,&quot;6C2&quot;</td>
</tr>
<tr>
<td>6C6</td>
<td>SETC</td>
<td>/<em>SYMBOL</em></td>
</tr>
<tr>
<td>6C7</td>
<td>SETC</td>
<td>&quot;6C1,UV<em>XYZ</em>&quot;</td>
</tr>
<tr>
<td>6C8</td>
<td>SETC</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>6C9</td>
<td>SETC</td>
<td>&quot;6C9&quot;,&quot;6C1&quot;(1:1)</td>
</tr>
</tbody>
</table>

SYMBOLIC PARAMETERS COULD HAVE BEEN USED IN ANY OF THE ABOVE
EXAMPLES IN PLACE OF CHARACTER VARIABLES. NOTICE THE USE OF
THE DOUBLE QUOTES IN 6C6 AND THE TRUNCATION OF THE STRING TO
6 CHARACTERS IN 6C7.

SUB-STRINGS MAY BE EXTRACTED BY A TECHNIQUE SIMILAR TO
The SUBSTR FUNCTION IN PL/I. A SUBSTRING IS:

CHARACTER EXPRESSION((6CSTART,6CLENGTH))

THAT IS A QUOTED STRING FOLLOWED BY A PAIR OF
PARENTHESES ENCLOSING 2 ARITHMETIC EXPRESSIONS. THE FIRST
ARITHMETIC EXPRESSION IS THE STARTING POSITION IN THE QUOTED
STRING, AND THE SECOND ARITHMETIC EXPRESSION IS THE LENGTH OF
NUMBER OF CHARACTERS TO EXTRACT. BOTH ARITHMETIC EXPRESSIONS
MUST BE PRESENT.

*6C0*(2,2)  BC
*6C0*(3,1)  C
*6C1*(1,2)  AE
*6C*(LINDEX+1)  THE SINGLE CHARACTER OF 6X DETERMINED BY THE
VALUE OF LINDEX
*6C0*"DEF"(2,2)  ASCEF NOTICE HOW CONCATENATION DEFINES THE
**602-COX**

**MACROS**

**SCOPE OF THE SUBSTRING**

```
*ABC0F1(2,2)     BC
*ABC1(2,1)*DEF*(2,1)  BE THE OPTIONAL PERIOD BETWEEN "DEF"
      MAY BE OMITTED.
```

**CHARACTER VARIABLES AND THE SUBSTRING NOTATION ARE**

**PARTICULARLY VALUABLE IN SCANNING ARGUMENTS FOR OCCURRENCES**

**OF CHARACTERS SUCH AS QUOTES, PARENTHESES, OR SPECIAL CHARACTERS.**

**CONSIDER THE PROBLEM IN GENERATING SS INSTRUCTIONS WITH A**

**LENGTH WHEN THE ARGUMENT CAN BE EITHER A VARIABLE OR IN BASE**

**DISPLACEMENT FORM.**

**MACRO**

```assembly
MACRO
    MOVE ALPHABETA,20,
    MVC ALPHA2(0),BETA

+    MOVE ALPHA+4,BETA+20
    MVC ALPHA+4(20),BETA

+    MOVE O(9),BETA+20
    MVC O(9)(20),BETA
```

**WHICH IS INCORRECT. WHAT WE WANT GENERATED IS**

```assembly
+    MVC O(20,9),BETA
```

**WE HAVE TO BE ABLE TO SCAN THE FIRST ARGUMENT FOR AN**

**OCURRENCE OF ( ). IF NO OCCURRENCE OF ( ) IS PRESENT THEN**

**THE ARGUMENT DOES NOT HAVE A BASE. IF ( ) DOES OCCUR THEN**

**EVERYTHING UP TO THE ( ) IS THE DISPLACEMENT AND EVERYTHING**

**BETWEEN THE ( ) IS THE BASE. WE WILL USE CHARACTER VARIABLES**

**AND SUBSTRINGS TO CHECK FOR OCCURRENCES OF ( ). WE WILL USE**

**ANOTHER CHARACTER VARIABLE TO BUILD UP THE DISPLACEMENT. WE**

**ALSO HAVE TO HAVE A WAY OF DETERMINING THE NUMBER OF CHARACTERS**

**IN THE ARGUMENT SO WE KNOW WHEN TO STOP THE SCAN.**

**KERSTM**

**IS A BUILTIN FUNCTION WHICH RETURNS THE NUMBER OF CHARACTERS**

**IN AN ARGUMENT. THE VALUE IS ARITHMETIC AND INCLUDES A COUNT**

**OF ALL PARENTHESES, QUOTES, AND EVERY OTHER CHARACTER INCLUDED**

**IN THE ARGUMENT.**

**MACRO**

```assembly
ENAME MVC ET0$FROM$LEN
LCLA ACCOUNT$PTR

* COUNT WILL BE A COUNT OF THE CHARACTERS IN THE ARGUMENT
* LPTR WILL SCAN ALONG THE CHARACTERS IN THE ARGUMENT
    LCLC EFIRST$REG
* GFIRST WILL BE THE DISPLACEMENT
+ GREG WILL BE THE BASE REGISTER
```
SET A COUNT TO THE NUMBER OF CHARACTERS

NEXT ANUP

GTPR SETA GPR +1 POINT TO THE NEXT CHARACTER

AIF (**GTO*(GTPR+1)) EQ **(*) BASE

IF A CHARACTER IS A LEFT PARENTHESIS

EFIRST SETC **FIRST* **GTO*(GTPR+1)

IF NOT A LP THEN BUILD UP EFIRST ONE CHAR AT A TIME

AIF (COUNT EQ GTPR) *NOBASE

IF WE REACH THE END OF GTO THEN THERE IS NO BASE REGISTER

AGO NEXT

*NOBASE ANUP

GNAME MVC GTO* (GLEN)* GFROM

* GENERATE THE ABOVE STATEMENT AND EXIT

MEXIT

BASE ANUP

GTPR SETA GTPR+1

AIF (**GTO*(GTPR+1) EQ *) *OK

SEARCHING FOR A RIGHT PARENTHESIS TO END THE BASE

GREG SETC **GREG* **GTO*(GTPR+1)

BUILD UP THE BASE REGISTER EXPRESSION

AGO BASE

OK ANUP

GNAME MVC EFIRST* (GLEN* GREG)* GFROM

SECOND FORM OF GENERATED STATEMENT

END

LET US EXAMINE THIS SEEMINGLY COMPLEX MACRO IN MORE DETAIL. THE FIRST PART OF THE MACRO DEFINES THE LOCAL VARIABLES AND ASSIGN COUNT TO THE NUMBER OF CHARACTERS IN THE FIRST ARGUMENT WHEN THE MACRO IS CALLED. THESE STATEMENTS ARE:

LCLA

LCLC

SETA

The next part of the macro is searching for an occurrence of a left parenthesis. We loop through from the anup to the agl incrementing the pointer GTPR by 1 each time. If we reach the end of the argument without encountering an occurrence of **(*) then we assume that the argument is not of the base-displacement form. In that case we branch to the generation of *NOBASE.* This occurs in the second AIF statement.

The first AIF statement looks complex but is not. **GTO*(GTPR+1) is the substring notation to pick out a single character of *GTO* to be compared with the character string **(*) of the entire expression is enclosed in parentheses of course. Expanded it looks like:

( **GTO*(GTPR+1) EQ **(*)

The SETC statement assigns to EFIRST the concatenation of **EFIRST* with the single character that is not a **(*) thus building up EFIRST to be the value of either the displacement on the variable. Remember that EFIRST is initialized to the NULL string. **GTO*(GTPR+1) will be the single character just
SCANNED IN THE AIF STATEMENT.

___ IF WE HAVE NO OCCURRENCE OF A LEFT PARENTHESIS, THEN IT IS ASSUMED THAT THERE IS NO BASE AND THE STATEMENT

MVC LTO,(LEN) &FRM

IS GENERATED. AFTER THAT AN MEXIT IS TAKEN. THE MEXIT ACTS JUST LIKE A STOP STATEMENT IN PL/I OR FORTRAN.

___ IF AN OCCURRENCE OF A LEFT PARENTHESIS OCCURS THEN WE ASSUME THERE IS A BASE REGISTER SPECIFIED. WE BRANCH TO BASE WHICH IS THE NEXT PART OF THE MACRO. HERE WE DO THE SAME AS IN THE LAST PART ONLY WE ARE SEARCHING FOR AN OCCURRENCE OF A RIGHT PARENTHESIS. UNTIL THAT OCCURS, WE BUILD UP THE BASE REGISTER IN GREG THE SAME WAY WE BUILT UP THE DISPLACEMENT IN &FIRST. WHEN THE OCCURRENCE OF *) OCCURS WE BRANCH TO THE GENERATION OR THE STATEMENT:

ENAME MVC &FIRST,(LEN),(REG) &FRM

WE NOTICE THAT THE PARAMETER ENAME OCCURS TWICE IN THIS MACRO. THERE IS NO PROBLEM SINCE ONLY ONE OF THE INSTRUCTIONS WILL BE GENERATED. EITHER THE :

ENAME MVC LTO,(LEN),&FRM

OR

ENAME MVC &FIRST,(LEN),(REG) &FRM

WILL BE GENERATED, BUT NOT BOTH.

BINARY OR BOOLEAN VARIABLES HAVE THE VALUE OF 0 FOR FALSE OR 1 FOR TRUE. A BINARY VARIABLE MAY BE ASSIGNED A VALUE BY THE EVALUATION OF A RELATIONAL EXPRESSION ENCLOSED IN PARENTHESES. THE VALUE OF THE EXPRESSION IS EITHER TRUE (1) OR FALSE (0). THAT VALUE IS THEN ASSIGNED TO THE BINARY VARIABLE. WHILE BINARY VARIABLES ARE NOT NECESSARY IN THE MACRO LANGUAGE, THEY MAKE MANY EXPRESSIONS EASIER TO INTERPRET. BINARY VARIABLES ARE PARTICULARLY USEFUL IN THE EVALUATION OF AIF STATEMENTS. TRUE OR FALSE VALUES MAY BE ASSIGNED TO BINARY VARIABLES AND THEN USED IN AIF EXPRESSIONS.

LCLB GB1,GB2,GB3,GB4

681 SETB 0 SET TO FALSE

681 SETB (0) SET TO FALSE USING OPTIONAL ( )

682 SETB (1) SET TO TRUE

682 SETB 1 SET TO TRUE

BINARY VARIABLES MAY ALSO BE ASSIGNED VALUES AS A RESULT OF LOGICAL COMPARISONS. BOTH OPERANDS MUST BE OF THE SAME TYPE, THAT IS BOTH MUST BE ARITHMETIC OR BOTH MUST BE CHARACTER EXPRESSIONS. LOGICAL EXPRESSIONS MAY BE FORMED BY USING THE OPERATORS AND, OR, NOT. THE EXPRESSION IS EVALUATED FROM LEFT TO RIGHT EXCEPT THAT AND IS EVALUATED BEFORE OR.

683 SETB (GB1 EQ 7)

684 SETB (GB1 OR GB2)

685 SETB (GB1 EQ 7 OR GB3)
VALUES OF SET VARIABLES MAY BE CONVERTED FROM ONE TYPE TO
ANOTHER IN A REASONSABLE WAY. ARITHMETIC VARIABLES MAY BE CONVERTED
TO CHARACTER STRINGS BY ENCLOSING THE VARIABLE IN QUOTES. THE
RESULT IS A CHARACTER REPRESENTATION OF THE NUMBER WITHOUT THE
SIGN. ARITHMETIC VARIABLES MAY BE USED AS BINARY VARIABLES
PROVIDED THE VALUE IS EITHER 0 OR 1. BINARY VARIABLES MAY BE
USED IN ARITHMETIC EXPRESSIONS OR CHARACTER EXPRESSIONS WITH NO
PROBLEM. THE VALUE WILL BE EITHER NUMERIC 0 OR 1 OR CHARACTER
0 OR 1. CHARACTER VARIABLES MAY BE USED IN ARITHMETIC OR BINARY
EXPRESSIONS PROVIDED THE CHARACTER VALUES ARE 1 TO 8 DIGITS OR
EITHER 0 OR 1. THE PROPER CONVERSION WILL TAKE PLACE.

LC1A  6A1.6A2
LC1B  6B1.6B2
LC1C  6C1.6C2
6A1  SETA  1
6A2  SETA  5
6B1  SETB  0
6B2  SETB  1
6C1  SETC  "5"
6C2  SETC  "5"

AIF (6A1 EQ 6A2).TRUE
AIF ('6A1' EQ '6A2').TRUE
6A2  SETA  6A1+6C2  RESULT IS ARITHMETIC 6
6C2  SETC  '6C1+6B1'  RESULT IS CHARACTER '10'
6A1  SETA  6C2-6A1  RESULT IS ARITHMETIC Y
AIF ('6A1' EQ '6B1').TRUE

ERROR MESSAGES AND COMMENTS MAY BE GENERATED BY A MACRO
BY USE OF THE MNOTE STATEMENT. THE FORM IS:

MNOTE CODE MESSAGE

THE CODE IS EITHER AN INTEGER BETWEEN 0 AND 255, A *, OR
OMITTED IN WHICH CASE IT IS ASSUMED TO BE 1. IF IT IS AN
INTEGER UNLIMITED THEN IT IS THE SEVERITY CODE JUST AS ANY
OTHER SEVERITY CODE PRODUCED BY THE ASSEMBLER. IF THE CODE
IS A *, THEN THE 'MESSAGE' IS PRINTED AS A COMMENT. MNOTES
ARE USEFUL FOR PRINTING DIAGNOSTICS ABOUT THE MACRO.

THE ATTRIBUTES OF ARGUMENTS OF A MACRO CALL MAY BE
TESTED IN A MACRO. THESE ATTRIBUTES ARE: LENGTH, TYPE, COUNT,
AND NUMBER.

LENGTH  L
NUMBER  N
COUNT  K
TYPE  T

LENGTH IS THE LENGTH ATTRIBUTE ASSIGNED TO THAT SYMBOL
BY THE ASSEMBLER. IT IS THE NUMBER OF BYTES IN THE DEFINITION
OF THE SYMBOL AND IS AN ARITHMETIC VALUE. L*SPARM MAY BE
USED IN ANY ARITHMETIC EXPRESSION.

NUMBER IS THE NUMBER OF ARGUMENTS PRESENT IN THE CALL OF
The macro \textit{it is computed as the number of commas plus 1}.

\texttt{MAC A\(\cdot\)B\(\cdot\)C N\#SYSLIST IS 3}
\texttt{MAC A\(\cdot\)A\(\cdot\)C N\#SYSLIST IS 2}
\texttt{MAC A\(\cdot\)B N\#SYSLIST IS 2}
\texttt{MAC (A\(\cdot\)B\(\cdot\)C)\(\cdot\)D N\#SYSLIST IS 2}
\texttt{N\#SYSLIST(1) IS 3}
\texttt{N\#SYSLIST(2) IS 1}
\texttt{MAC A\(\cdot\)A\(\cdot\)C N\#SYSLIST(2) IS 0}

The \texttt{number} function is useful in a macro that can have a variable number of arguments. Consider the following macro which generates full word constants:

\begin{verbatim}
MACRO
  FCNS
  LCLA E1
  .LOOP    .ANOP
    AIF (E1 EQ N\#SYSLIST) DONE
    SETA E1++
  FESYSLIST(E1) DC F\#SYSLIST(E1)*
    AGO .LOOP
  MEND
  FCNS 1,5,20
  +F1 DC F\#1*
  +F5 DC F\#5*
  +F20 DC F\#20*
\end{verbatim}

The \texttt{count} function returns an arithmetic value that is equal to the number of characters in the argument. We have already seen a use for the \texttt{count} function.

The \texttt{type} function returns a single character value which is a letter. The type is how the symbol is defined: full word constant, half word constant, address constant, etc. The value may be used in any character expression.

\begin{verbatim}
T\#PARAM
\end{verbatim}

The values return and their meanings are:

A ADDRESS CONSTANT
B DINARY CONSTANT
C CHARACTER CONSTANT
D DOUBLE WORD
F FULL WORD
H HALF WORD
I MACHINE INSTRUCTION
J CONTROL SECTION NAME
M MACRO INSTRUCTION
N SELF DEFINING TERM
O OMITTED OPERAND
P PACKED DECIMAL CONSTANT
S BASE DISPLACEMENT CONSTANT
U UNDEFINED
X HEXADECIMAL CONSTANT
MACRO DECIMAL CONSTANT

The type function is useful in determining if the proper argument has been passed to the macro, or in generating the correct code for the type of argument.

This macro generates either full word instruction or half word instruction depending on the type of argument in the macro call.

MACRO
ADD       EREG,LA,DL,CL
LCLC      LT
AIF       (T*LA NE T*LB)*ERROR1
AIF       (T*LA NE T*LC)*ERROR1
AIF       (T*LA EQ *H* OR T*LA EQ *F*),OK
MNOTE     ERROR, "ARGUMENTS NOT FULL OR HALF WORD CONSTANTS!"
MEXIT
OK
AIF       (T*EA EQ *F*),GEN
LT
SETC      T*EA
*GEN
ANUP
LJ
AIF       EREG,EA
ADT       EREG,EB
ST&I      EREG,EC
MEXIT
ERROR1
ANUP
MNOTE 6, "ARGUMENTS NOT THE SAME TYPE!"
MEND

Notice that when T*EA is used in a character expression it is not enclosed in quotes. When it is used as an operand for SETC it must be used alone and not enclosed in quotes, if it were enclosed in quotes then it would not be a function but a character string.