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 ASSEMELY LANGUAGE WE HAVE, SO FAR, TWO TYPES OF INSTRUCTIONS. MACHINE LANGUAGE INSTRUCTIONS WHICH, WHEN PROCESSED BY THE ASSEMELER, PRODUCE MACHINE CODE. EXAMPLE. L 3,ALPHA A 4,BETA MVC D(4,6),S(10). A SECOND TYPE ARE THE ASSEMBLER INSTRUCTIONS, WHICH WHEN PROCESSED BY THE ASSEMBLER, SOMETIMES PRODUCE CONSTANTS AND SOMETIMES JUST INFORMS 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 ORG INFORM THE ASSEMBER 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, DCB, READ, WRITE, DCBD, SNAP, AND PUTX. SUPERVISOR SERVICES OF MACROS ARE FOR CONTROL OF YOUR PROGRAM AND STORAGE. THEY INCLUDE: SAVE, RETURN, TIME, GETMAIN, SPIE, STAE, AND ABEND. 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 BALR 14,15 INSTRUCTION. REG 15 CONTAINS THE ENTRY POINT (STARTING ADDRESS) OF YOUR PROGRAM, AND REG 14 CONTAINS THE RETURN ADDRESS TO US. REG 0 AND 1 MAY CONTAIN PARAMETERS OR POINT TO PARAMETER LISTS FOR YOUR PROGRAM. IN ADDITION REG 13 CONTAINS THE ADDRESS OF AN 18 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 BR 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 BR 14 INSTRUCTION FOR YOU.

```
         SAVE      (14,12)
         +STM      14,12,12(15)
```

```
        RETURN  (14,12)
        +LM     14,12,12(13)
        +BR     14
```

THE GENERATED INSTRUCTIONS STM, 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+4 GAMMA-6 DELTA(7)
RX TYPE	ANY ADDRESS THAT CAN BE USED IN AN RX TYPE INSTRUCTION D(4) B(3,2) OR ANY OF THE TYPE SYM
DEC DIG	ANY DECIMAL DIGIT 2 4 12 14
REGISTER	ALWAYS CODED IN PARENTHESES
	(2-12) 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.

MACROS 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 16. THESE COLUMNS AND RULES MUST BE STRICTLY ADHEARED TO.

IN ADDITION TO THE IBM 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
        MAC
        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 HAD ANOTHER MACRO NAMED MAC1:

```
*       MACRO
        MAC1
        INST3
*       MAC
        INST4
        MEND
```

THEN THE EXPANSION OF:

```
        INST1
        MAC1
        INST2
```

WOULD BE:

```
        INST1
```

```
+    INST3
+    LOAD
+    ADD
+    STORE
+    INST4
     INST2
```

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
     .
     DSECT
     .          FIRST DUMMY SECTION
     DSECT
     .          SECOND DUMMY SECTION
     END        END STATEMENT
```

DURING THE ASSEMBLY PROCESS THE SOURCE CODE IS FIRST PASSED TO A PREPROCESSOR CALLED A MACRO PROCESSOR. THE MACRO PROCESSOR EXAMINES EACH STATEMET. 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

OF SEARCHING 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	&AREA	&LOC
&LOC1	&SYM	&INDEX
&LABEL	&NAME	&LIST

SEQUENCE SYMBOLS ARE DEFINED IN THE SAME MANNER EXCEPT THAT THE FIRST CHARACTER IS A PERIOD. VALID EXAMPLES ARE:

.DONE	.END	.MORE
.NEXT	.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 PROTOTYPE 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:

```
EX1     L     5,ALPHA
        A     5,BETA
        ST    5,GAMMA
```

THE MACRO COULD BE CODED:

```
        MACRO
&NAME   ADDS  &REG,&LOC1,&LOC2,&LOC3
&NAME   L     &REG,&LOC1
        A     &REG,&LOC2
        ST    &REG,&LOC3
```

```
+          BE        OUT0001                 BRANCH OUT
+          LA        4,1(4)                  ADD 1 TO REGISTER
+          B         SC0002                  BRANCH BACK
+OUT0002   EQU       *
```

WE CAN SEE THAT DIFFERENT CALLS OF THE SAME MACRO PRODUCE DIFFERENT LABELS. HAD WE NOT HAD THE FACILITY OF &SYSNDX 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 MACRO, BUT IF WE SPECIFY A REGISTER AND ONLY 2 LOCATIONS THEN WE HAVE TO USE THE ADDNS MACRO. 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).SEQSYM
```

THE (CONDITION) IS EVALUATED. IF IT IS TRUE THEN A BRANCH IS TAKEN TO THE SEQUENCE SYMBOL .SEQSYM AND IF THE CONDITION IS FALSE THE NEXT INSTRUCTION IS TAKEN. THIS ACTS JUST LIKE AN IF (CONDITION) GO TO INSTRUCTION IN PL/1 OR FORTRAN. THE CONDITION IS WRITTEN AS:

(EXPRESSION1 RELATION EXPRESSION2)

THE RELATIONS ARE:

EQ	EQUAL
NE	NOT EQUAL
LT	LESS THAN
LE	LESS THAN OR EQUAL
GT	GREATER THAN
GE	GREATER THAN OR EQUAL

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

```
          MACRO
&NAME     ADD       &REG,&LOC1,&LOC2,&LOC3
&NAME     L         &REG,&LOC1
          A         &REG,&LOC2
          AIF       ('&LOC3' EQ '').DONE
          ST        &REG,&LOC3
```

```
.DONE   MEND
```

THE AIF STATEMENT TESTS WHETHER THE 4TH ARGUMENT &LOC3 IS NULL. THE NULL STRING IS SPECIFIED AS TWO SINGLE QUOTES WITH NO SPACE BETWEEN THEM. SINCE NULL '' IS A CHARACTER EXPRESSION WE NEED THE SINGLE QUOTES AROUND '&LOC3'. IF THE CONDITION IS TRUE, THAT IS '&LOC3' IS NULL THEN WE SKIP THE STORE INSTRUCTION AND BRANCH TO THE MEND. IF IT IS FALSE THEN WE GENERATE THE STORE INSTRUCTION AND THEN PERFORM THE MEND. THE SEQUENCE SYMBOL .DONE IS WRITTEN IN THE LABEL FIELD OF THE MEND STATEMENT.

```
        ADD     6,A,B,C
+       L       6,A
+       A       6,B
+       ST      6,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 &A IS A SYMBOLIC PARAMETER, THEN &A(1) REFERS TO THE FIRST ARGUMENT OF THE SUBLIST WHILE &A(2) REFERS TO THE SECOND ARGUMENT OF THE SUBLIST.

CONSIDER THE FOLLOWING MACRO WHICH ADDS THREE LOCATIONS AND STORES THE RESULT IN SUM:

```
        MACRO
&NAME   ADD3    &LOC,&REG,&SUM
.&NAME  L       &REG,&LOC(1)
        A       &REG,&LOC(2)
        A       &REG,&LOC(3)
        ST      &REG,&SUM
        MEND

        ADD3    (A,B,C),5,ALPHA
+       L       5,A
+       A       5,B
+       A       5,C
+       ST      5,ALPHA
```

THE ARGUMENT THAT CORRESPONDS TO THE SYMBOLIC PARAMETER &LOC CONSISTS OF THREE ARGUMENTS IN A SUBLIST ENCLOSED IN PARENTHESES. &LOC(1) REFERS TO THE FIRST ARGUMENT IN THE SUBLIST OR A, &LOC(2) REFERS TO THE SECOND ARGUMENT IN THE

SUBLIST OR 5. WHILE &LOC(3) REFERS TO THE THIRD ARGUMENT OF THE SUBLIST OR C.

WE NOW SEE THE REASON WHY A PERIOD MUST BE USED WHEN CONCATENATING A VARIABLE SYMBOL AND A LEFT PARENTHESIS. &LOC(2) REFERS TO AN ARGUMENT IN THE SUBLIST, WHILE &LOC.(2) REFERS TO THE ENTIRE LIST OF ARGUMENTS CONCATENATED WITH (2). THIS WOULD RESULT IN (A,B,C) (2) BEING GENERATED.

```
         MACRO
         ADDBAD   &LOC,&REG,&SUM
         L        &REG,&LOC.(1)
         .
         .

         ADDBAD   (A,B,C),5,ALPHA
*        L        5,(A,B,C)(1)
         .
         .
```

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)      5
&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    ADD3     &LOC,&REG,&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)
         MEND
```

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 ALLOW 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 THE 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 HAVE 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 MACRO.

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:

LCLA	LOCAL ARITHMETIC INITIAL VALUE 0
LCLB	LOCAL BINARY OR BOOLEAN INITIAL VALUE 0
LCLC	LOCAL CHARACTER INITIAL VALUE NULL ''
GBLA	GLOBAL ARITHMETIC INITIAL VALUE 0
GBLB	GLOBAL BINARY OR BOOLEAN INITIAL VALUE 0
GBLC	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 &SYS.

LCLA	&I	DEFINES A LOCAL ARITHMETIC VARIABLE
LCLB	&T,&F	DEFINES 2 LOCAL BINARY OR BOOLEAN VARIABLES
LCLC	&TYPE,&KIND	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:

&VARIABLE	SETA	ARITHMETIC EXPRESSION
&VARIABLE	SETB	(BOOLEAN OR BINARY EXPRESSION)
&VARIABLE	SETC	CHARACTER EXPRESSION

&I	SETA	1	ASSIGN 1 TO &I
&I	SETA	&I+1	INCREMENT VARIABLE &I
&I	SETA	&I-1	DECREMENT &I
&J	SETA	&I+4	ADD 4 TO &I AND ASSIGN TO &J
&LEN	SETA	L'&PARM	ASSIGN TO &LEN THE LENGTH ATTRIBUTE OF PARAMETER &PARM

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 SUBRACTION. PARENTHESES MAY BE USED TO REDEFINE THE ORDER OF EVALUATION. A MAXIMUM OF 16 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
&NAME    LOST    &FROM,&TO
         LCLA    &A,&B
         LCLA    &C,&D
&A       SETA    10
&B       SETA    12
&C       SETA    &A-&B
&D       SETA    &A+&C
&NAME    L       2,&FROM&C
         ST      2,&TO&D
         MEND

HERE     LOST    ALPHA,BETA
+HERE    L       2,ALPHA2
+        ST      2,BETA8
```

THE VALUES ARE DETERMINED AS FOLLOWS: &A AND &B ARE SET TO THE VALUES 10 AND 12 RESPECTIVELY IN THE SETA STATEMENTS. &C IS ASSIGNED THE VALUE OF -2 AS A RESULT OF THE ARITHMETIC EXPRESSION &A-&B. &D IS ASSIGNED THE VALUE +8 AS A RESULT OF THE ARITHMETIC EXPRESSION &A+&C. ALPHA2 IS A RESULT OF THE CONCATENATION OF &FROM&C. &FROM HAS THE VALUE ALPHA AND &C IS CONVERTED TO AN UNSIGNED INTEGER 2 WITH LEADING ZEROS STRIPPED OFF. BETA8 IS A RESULT OF THE CONCATENATION OF &TO&D. &TO HAS THE VALUE BETA AND &D 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 R,R INSTRUCTION FOR EACH REGISTER IN THE RANGE. A WRAP-AROUND FEATURE HAS BEEN BUILT INTO THE MACRO.

```
         CLREG   1,2
+        SR      14,14
+        SR      15,15
+        SR      0,0
+        SR      1,1
+        SR      2,2

         MACRO
         CLREG   &R1,&R2
         LCLA    &COUNT
&COUNT   SETA    &R1
.AGAIN   ANOP
```

```
         SR    &COUNT,&COUNT
         AIF   (&COUNT EQ &R2).DONE
         AIF   (&COUNT EQ 15).RESET
.COUNT   SETA  &COUNT+1
         AGO   .AGAIN
.RESET   ANOP
&COUNT   SETA  0
         AGO   .AGAIN
.DONE    MEND
```

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

```
.AGAIN   EQU   *
```

BY GIVING US A SEQUENCE SYMBOL TO BRANCH TO WHEN WE ENCOUNTER THE
AGO .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 AGO 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 PROPOGATE THE BLANK BY AN
MVC WITH A LENGTH OF 1 LESS THAN THE LENGTH OF THE AREA. LET
US WRITE A MACRO TO PERFORM THAT FUNCTION.

```
         MACRO
.NAME    CLEAR &AREA,&LEN
         LCLA  &COUNT
.NAME    MVI   &AREA,C' '      CLEAR THE FIRST BYTE
.COUNT   SETA  &LEN-1          SET THE LENGTH MINUS 1
         MVC   &AREA+1(&COUNT),&AREA   CLEAR THE REST OF THE AREA
         MEND

         CLEAR ALPHA,80
         MVI   ALPHA,C' '              CLEAR THE FIRST BYTE
         MVC   ALPHA+1(79),ALPHA       CLEAR THE REST OF AREA
```

CHARACTER EXPRESSIONS ARE DEFINED AS UP TO 255 CHARACTERS
ENCLOSED 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 LCLC STATEMENT. SEVERAL VARIABLES MAY BE DEFINED WITH ONE LCLC 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'&SYM ATTRIBUTE. I' 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)
3) SUBSTRING NOTATION
4) CONCATENATION OF 2) AND 3) ABOVE.

```
         LCLC   &C1,&C2,&C3,&C5,&C6,&C7,&C8,&C9
&C1      SETC   'ABCD'          ASSIGNS THE STRING ABCD TO &C1
&C2      SETC   'AB'.'CD'       CONCATENATES AB WITH CD AND
                                ASSIGNS ABCD TO &C2
&C3      SETC   '&C1'           ASSIGNS THE VALUE OF &C1 WHICH IS
                                ABCD TO &C3
&C4      SETC   '&C1&C2'        CONCATENATE &C1 WITH &C2 AND ASSIGNS
                                ABCDABCD TO &C4
&C5      SETC   '&C1'.'&C2'     SAME AS &C4
&C6      SETC   'L''SYMBOL'     ASSIGNS L'SYMBOL TO &C6
&C7      SETC   '&C1.UV#XYZ'    ASSIGNS ABCDUVWX TO &C7
&C8      SETC   ''              ASSIGNS THE NULL STRING
&C9      SETC   '&C9'.'&C1'(1,1) BUILDS UP CHARACTERS IN &C9
```

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 &C6 AND THE TRUNCATION OF THE STRING TO 8 CHARACTERS IN &C7.

SUB-STRINGS MAY BE EXTRACTED BY A TECHNIQUE SIMILAR TO THE SUBSTR FUNCTION IN PL/1. A SUBSTRING IS:

```
*   'CHARACTER EXPRESSION'(&START,&LENGTH)
```

*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.

'ABCD'(2,2)	BC
'ABCD'(3,1)	C
'&C1'(1,2)	AB
'&X'(&INDEX,1)	THE SINGLE CHARACTER OF &X DETERMINED BY THE VALUE OF &INDEX
'ABC'.'DEF'(2,2)	BCEF NOTICE HOW CONCATENATION DEFINES THE

```
                        SCOPE OF THE SUBSTRING
*ABCDEF*(2,2)           BC
*ABC*(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
         MOVE    &TO,&FROM,&LEN
         MVC     &TO.(&LEN),&FROM
         MEND

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

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

         MOVE    0(9),BETA,20
+        MVC     0(9)(20),BETA
```

WHICH IS INCORRECT. WHAT WE WANT GENERATED IS

```
+        MVC     0(20,9),BETA
```

WE HAVE TO BE ABLE TO SCAN THE FIRST ARGUMENT FOR AN OCCURRENCE 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.

```
     K'&SYM
```

IS A BUILT IN FUCTION 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
&NAME    MOVE    &TO,&FROM,&LEN
         LCLA    &COUNT,&PTR
.*   &COUNT WILL BE A COUNT OF THE CHARACTERS IN THE ARGUMENT
.*   &PTR WILL SCAN ALONG THE CHARACTERS IN THE ARGUMENT
         LCLC    &FIRST,&REG
.*   &FIRST WILL BE THE DISPLACEMENT
.*   &REG WILL BE THE BASE REGISTER
```

```
&COUNT   SETA    K'&TO          SET &COUNT TO THE NUMBER OF CHARACTERS
.NEXT    ANOP
&PTR     SETA    &PTR+1         POINT TO THE NEXT CHARACTER
         AIF     ('&TO'(&PTR,1) EQ '(').BASE
.*   BRANCH TO .BASE IF A CHARACTER IS A LEFT PARENTHESIS
&FIRST   SETC    '&FIRST'.'&TO'(&PTR,1)
.*   IF NOT A LP THEN BUILD UP &FIRST ONE CHAR AT A TIME
         AIF     (&COUNT EQ &PTR).NOBASE
.*   IF WE REACH THE END OF &TO THEN THERE IS NO BASE REGISTER
         AGO     .NEXT
.NOBASE  ANOP
&NAME    MVC     &TO.(&LEN),&FROM
.*   GENERATE THE ABOVE STATEMENT AND EXIT
         MEXIT
.BASE    ANOP
&PTR     SETA    &PTR+1
         AIF     ('&TO'(&PTR,1) EQ ')').OK
.*   SEARCHING FOR A RIGHT PARENTHESIS TO END THE BASE
&REG     SETC    '&REG'.'&TO'(&PTR,1)
.*   BUILD UP THE BASE REGISTER EXPRESSION
         AGO     .BASE
.OK      ANOP
&NAME    MVC     &FIRST.(&LEN.&REG).&FROM
.*   SECOND FORM OF GENERATED STATEMENT
         MEND
```

LET US EXAMINE THIS SEEMINGLY COMPLEX MACRO IN MORE DETAIL. THE FIRST PART OF THE MACRO DEFINES THE LOCAL VARIABLES AND ASSIGNS &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 ANOP TO THE AGO INCREMENTING THE POINTER &PTR 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. '&TO'(&PTR,1) IS THE SUBSTRING NOTATION TO PICK OUT A SINGLE CHARACTER OF '&TO' TO BE COMPARED WITH THE CHARACTER STRING '('. THE ENTIRE EXPRESSION IS ENCLOSED IN PARENTHESES OF COURSE. EXPANDED IT LOOKS LIKE:

```
(  '&TO'(&PTR,1)    EQ     '('     )
```

THE SETC STATEMENT ASSIGNS TO &FIRST THE CONCATENATION OF '&FIRST' WITH THE SINGLE CHARACTER THAT IS NOT A '(' THUS BUILDING UP &FIRST TO BE THE VALUE OF EITHER THE DISPLACEMENT OR THE VARIABLE. REMEMBER THAT &FIRST IS INITIALIZED TO THE NULL STRING. '&TO'(&PTR,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     &TO.(&LEN),&FROM
```

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 &REG THE SAME WAY WE BUILT UP THE DISPLACEMENT IN &FIRST. WHEN THE OCCURRENCE OF ')' OCCURS WE BRANCH TO THE GENERATION OR THE STATEMENT:

```
&NAME       MVC       &FIRST.(&LEN,&REG),&FROM
```

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

```
&NAME       MVC       &TO.(&LEN),&FROM              OR
&NAME       MVC       &FIRST.(&LEN,&REG),&FROM
```

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    &B1,&B2,&B3,&B4
&B1         SETB    0           SET TO FALSE
&B1         SETB    (0)         SET TO FALSE USING OPTIONAL ( )
&B2         SETB    (1)         SET TO TRUE
&B2         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.

```
&B3         SETB    (&REG EQ 7)
&B4         SETB    (&B1 OR &B2)
&B5         SETB    (&REG EQ 7 OR &B3)
```

VALUES OF SET VARIABLES MAY BE CONVERTED FROM ONE TYPE TO ANOTHER IN A REASONABLE 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.

```
         LCLA    &A1,&A2
         LCLB    &B1,&B2
         LCLC    &C1,&C2
&A1      SETA    1
&A2      SETA    5
&B1      SETB    0
&B2      SETB    1
&C1      SETC    '1'
&C2      SETC    '5'
         AIF     (&A1 EQ &B2).TRUE
         AIF     ('&A1' EQ '&C1').TRUE
&A2      SETA    &A1+&C2          RESULT IS ARITHMETIC 6
&C2      SETC    '&C1&B1'         RESULT IS CHARACTER '10'
&A1      SETA    &C2-&A1          RESULT IS ARITHMETIC 9
         AIF     ('&A1' EQ '9').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 OR OMITTED 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'&PARM MAY BE USED IN ANY ARITHMETIC EXPRESSION.

NUMBER IS THE NUMBER OF ARGUMENTS PRESENT IN THE CALL OF

THE MACRO. IT IS COMPUTED AS THE NUMBER OF COMMAS PLUS 1.

```
      MAC   A,B,C          N'&SYSLIST  IS 3
      MAC   A,,C           N'&SYSLIST  IS 3
      MAC   A,B            N'&SYSLIST  IS 2
      MAC   (A,B,C),D      N'&SYSLIST  IS 2
                           N'&SYSLIST(1) IS 3
                           N'&SYSLIST(2) IS 1
      MAC   A,,C           N'&SYSLIST(2) IS 0
```

THE NUMBER FUNCTION IS USEFUL IN A MACRO THAT CAN HAVE A VARIABLE NUMBER OF ARGUMENTS. CONSIDER THE FOLLOWING MACRO WHICH GENERATES FULL WORD CONSTANTS.

```
              MACRO
              FCONS
              LCLA    &I
.LOOP         ANOP
              AIF     (&I EQ N'&SYSLIST).DONE
&I            SETA    &I+1
F&SYSLIST(&I) DC    F'&SYSLIST(&I)'
              AGO     .LOOP
              MEND

              FCONS   1,5,20
+F1           DC      F'1'
+F5           DC      F'5'
+F20          DC      F'20'
```

THE 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 COUNT FUNCTION.

THE 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.

```
      T'&PARM
```

THE VALUES RETURN AND THEIR MEANINGS ARE:

A	ADDRESS CONSTANT
B	BINARY 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	HEXIDECIMAL CONSTANT

```
Z       ZONED 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         &REG,&A,&B,&C
            LCLC        &T
            AIF         (T'&A  NE  T'&B).ERROR1
            AIF         (T'&A  NE  T'&C).ERROR1
            AIF         (T'&A  EQ  'H' OR T'&A EQ 'F').OK
            MNOTE       8,'ARGUMENTS NOT FULL OR HALF WORD CONSTANTS'
            MEXIT
OK.         ANOP
            AIF         (T'&A  EQ  'F').GEN
&T          SETC        T'&A
.GEN        ANOP
            L&T         &REG,&A
            A&T         &REG,&B
            ST&T        &REG,&C
            MEXIT
.ERROR1     ANOP
            MNOTE 8,'ARGUMENTS NOT THE SAME TYPE'
            MEND

        NOTICE THAT WHEN T'&A 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.
```