# MOL620
# A MACHINE ORIENTED LANGUAGE AND LANGUAGE COMPILER FOR THE VARIAN DATA 620/I

MARSHA DRAPKIN HOPWOOD
GREGORY L. HOPWOOD

v.1, 1971

TECHNICAL REPORT NO. 1
SEPTEMBER 1971*
DEPARTMENT OF INFORMATION AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA, IRVINE

*THIS IS A MAJOR REVISION OF TECHNICAL REPORT NO. 1, APRIL 1969, BY MARSHA A. DRAPKIN. THE WORK DESCRIBED IN THAT REPORT WAS SUPPORTED BY A GRANT FROM THE CARNEGIE CORPORATION OF NEW YORK.

Z
699
C3
no. 1

3-12-73 1, 5, 8-10, 12-16, 18, 22

TABLE OF CONTENTS.

Reid.

# MOL620: A MACHINE ORIENTED LANGUAGE AND LANGUAGE COMPILER FOR THE VARIAN DATA 620/I

## INTRODUCTION.

THE PURPOSE OF MOL620 IS TO PROVIDE A HIGH LEVEL ALGOL-LIKE LANGUAGE WITH WHICH ONE CAN CONVENIENTLY REPRESENT ALGORITHMS FOR EXECUTION ON THE VARIAN DATA 620/I, A 16-BIT MINI-COMPUTER. THE ABILITY TO EXPRESS ALGORITHMS IN THE BASIC ASSEMBLY LANGUAGE OF THE MACHINE IS ALSO PROVIDED.

A HIGH LEVEL LANGUAGE IS ADVANTAGEOUS SINCE PROGRAMS CAN BE MORE EASILY WRITTEN AND DEBUGGED THAN IN ASSEMBLY LANGUAGE. THE PROGRAMMER AVOIDS MANY ERRORS HE WOULD NORMALLY MAKE IN THE COURSE OF WRITING A PROGRAM IN ASSEMBLY LANGUAGE BCAUSE MOL620 PROVIDES THE FACILITIES FOR DESCRIBING A COMPUTATION WITH FAR FEWER SYMBOLS AND THE LOGICAL FLOW OF THE PROGRAM IS NATURALLY DESCRIBED BY THE SYNTAX OF THE PROGRAM ITSELF.

THE MOL620 TRANSLATOR ACCEPTS AS INPUT A PROGRAM WRITTEN IN THE MOL620 LANGUAGE AND OUTPUTS A PROGRAM TO BE ASSEMBLED BY THE VARIAN DATA 620/I ASSEMBLER.

THE CURRENT VERSION OF THE MOL620 COMPILER IS WRITTEN IN TREE-META AND EXECUTES ON THE DIGITAL EQUIPMENT CORPORATION PDP-10 COMPUTER UNDER THE STANDARD MONITOR.

THE FIRST VERSION OF THE MOL620 COMPILER WAS WRITTEN IN FORTRAN IV-H AND EXECUTED ON AN IBM 360/50 UNDER OS.

SYNTAX OF MOL620 LANGUAGE.

THE SYNTAX IS DESCRIBED USING A METALANGUAGE SIMILAR TO BACKUS NAUR FORM.

THE CATEGORY NAME APPEARS ON THE LEFT OF THE EQUAL SIGN AND THE SYNTACTIC ALTERNATIVES APPEAR ON THE RIGHT. LITERALS ARE SURROUNDED BY QUOTES (") AND CATEGORY NAMES ARE NOT. FOR EXAMPLE "DONE" IS A LITERAL STRING, UNIT IS A CATEGORY NAME.

ALTERNATIVES IN A DEFINITION ARE SEPARATED BY A SLASH (/) INSTEAD OF THE CUSTOMARY VERTICAL BAR. A DEFINITION IS TERMINATED BY A SEMI-COLON (;). ALTERNATIVES ARE FACTORED BY THE USE OF PARENTHESES. FOR EXAMPLE, THE DEFINITION

```
        ASSIGNS = DESIG ("←" / "=") EXP ;
```

IS EQUIVALENT TO

```
        ASSIGNS = DESIG "←" EXP / DESIG "=" EXP ;
```

ITERATION OF A COMPONENT IN A RULE IS INDICATED BY THE M$N(...) CONSTRUCT, WHICH SAYS THAT THE COMPONENT IN PARENTHESES IS TO OCCUR AT LEAST M AND AT MOST N TIMES. IF M DOES NOT APPEAR, ZERO IS ASSUMED. IF N DOES NOT APPEAR, INFINITY IS ASSUMED. FOR EXAMPLE, THE DEFINITION

```
        DESIG = STOLOC $("," STOLOC) ;
```

SAYS THAT A DESIG IS A STOLOC FOLLOWED BY ZERO OR MORE OCCURRENCES OF "," STOLOC, I.E. A DESIG IS A STOLOC OR STOLOC,STOLOC,...,STOLOC.

AN ALTERNATIVE ENCLOSED IN ANGLE BRACKETS (<>) IS AN INFORMAL DEFINITION OF THE ALTERNATIVE.

```
PROGRAM = IDORG $(UNIT) "DONE."
IDORG   = $(DEC / COMMENT) (IDEN "ORG" VALUE ";" / EMPTY);
UNIT    = PROC / 1$(DEC / COMMENT) ;
PROC    = PROCHEAD FORMPART ";" (COMSTMS / EMPTY) "ENDP" ";" ;
PROCHEAD= ("PROCEDURE" / "PROC") IDEN ;
FORMPART= "(" (IDEN / EMPTY) ("," (IDEN / EMPTY) / EMPTY)
            ("," (IDEN / EMPTY) / EMPTY) ")" /
            EMPTY;
COMSTMS = 1$(DEC / STMNT / COMMENT);
COMMENT = DAS ";"/<STRING OF CHARACTERS ENCLOSED IN PERCENT SIGNS (%)>;
DEC     = DECLARE / ASSDEC;
ASSDEC  = ("SET" / "EQU") AITEM $("," AITEM) ;
AITEM   = IDEN "=" VALUE ;
DECLARE = ("DECLARE" / "DECL") ITEM $("," ITEM) ";" ;
ITEM    = IDEN ("[" (VALUE ":" / EMPTY) VALUE "]" / EMPTY)
            ("=" VALUE / EMPTY) ;
STMNT   = (LABELID / EMPTY) USTMNT ";" ;
LABELID = IDEN ":" ;
USTMNT  = IFS / FORS / WHILES / BLOCKS / ASSIGNS / NULLS /
            GOTOS / CALLS / RETURNS / BUMPS / STOPS / DAS ;
```

```
IFS      = "IF" EXP "THEN" USTMNT ("ELSE" USTMNT / EMPTY) ;
FORS     = "FOR" STOLOC ("←" / "=")  EXP
            (("STEP" EXP / EMPTY) ("UNTIL" / "WHILE") EXP)
            "DO" USTMNT ;
WHILES   = "WHILE" EXP "DO" USTMNT ;
ASSIGNS  = DESIG ("←" / "=") EXP ;
DESIG    = STOLOC $("," STOLOC) ;
NULLS    = "NULL" ;
BLOCKS   = "BEGIN" BODY "END" ;
BODY     = COMSTMS / EMPTY ;
GOTOS    = "GO" "TO" JMPLOC ;
CALLS    = "CALL" JMPLOC (ARGLIST / EMPTY) ;
RETURNS  = "RETURN" (ARGLIST / EMPTY) ("FROM" JMPLOC / EMPTY) ;
ARGLIST  = "(" (EXP / EMPTY)
            ("," (EXP / EMPTY) / EMPTY)
            ("," (EXP / EMPTY) / EMPTY) ")" ;
DAS      = <STRING OF CHARACTERS ENCLOSED IN QUOTES (")> ;
BUMPS    = "BUMP" STOLOC $("," STOLOC) ;
STOPS    = "STOP" / "HALT" ;
EXP      = INTRSECT $(("XOR" / "OR") INTRSECT) ;
INTRSECT= NEGATION $("AND" NEGATION) ;
NEGATION= "NOT" NEGATION / RELATION ;
RELATION= SUM $((">=" / ">" / "=" / "<=" / "<" / "#") SUM) ;
SUM      = PRODUCT $(("+" / "-") PRODUCT) ;
PRODUCT  = FACTOR $(("*" / "/") FACTOR) ;
FACTOR   = BITAND $(("BOR" / "BXR") BITAND) ;
BITAND   = SHIFT $("BAND" SHIFT) ;
SHIFT    = PRIMARY $( """" IDEN """" VALUE) ;
PRIMARY  = IDEN ("[" EXP "]" / EMPTY) /
             JMPLOC ARGLIST /
           "(" (REGISTER / SWITCH / EXP) ")" /
           CONSTANT /
           "[" EXP "]" /
           ("+" / "-") PRIMARY /
           "@" IDEN ;
CONSTANT= ("+" / "-" / EMPTY) NUMBER /
           <STRING IN QUOTES (")> /
           <ONE OR TWO CHARACTERS IN PRIMES (')> /
           "[" (IDEN / NUMBER) "]" ;
REGISTER= "AR" / "BR" / "XR" / "OF" ;
SWITCH   = "SS1" / "SS2" / "SS3" ;
STOLOC   = IDEN ("[" EXP "]" / EMPTY) /
           "(" REGISTER ")" / CONSTANT / "[" EXP "]" ;
JMPLOC   = IDEN ("[" EXP "]" / EMPTY) /
            CONSTANT / "[" EXP "]" ;
VALUE    = IDEN / CONSTANT / "-" IDEN ;
IDEN     = <STRING OF LETTERS AND DIGITS STARTING WITH LETTER> ;
NUMBER   = <STRING OF DIGITS> ;
EMPTY    = <NULL STRING; NOTHING> ;
```

## SEMANTICS OF THE MOL620 LANGUAGE.

### ASSIGNMENT STATEMENT.

AN ASSIGNMENT STATEMENT CONSISTS OF THREE PARTS:
(1) A LIST OF STORAGE LOCATIONS,
(2) AN ASSIGNMENT OPERATOR (EITHER '←' OR '='),
(3) AN EXPRESSION TO BE EVALUATED AND STORED IN THE LOCATIONS INDICATED BY (1).

EXAMPLES:

```
        A←B;
        C,D,E←ALPHA/BETA;
        LINK[X],X←LINK[Y];
        (XR)←F(1,3,A-B)-Q32;
        [Y+1] = (N-2)*7;
        T=-1;
        REACT←A OR B AND NOT C;
```

THE LIST OF STORAGE LOCATIONS IS SEPARATED BY COMMAS. STORAGE OF THE EXPRESSION VALUE AND EVALUATION OF ANY EXPRESSION ON THE LEFT OF THE ASSIGNMENT IS DONE RIGHT TO LEFT. THUS, THE STATEMENT

```
        LINK[X],X←LINK[Y];
```

IS EQUIVALENT TO

```
        T←LINK[Y]; X←T; LINK[X]←T;
```

FOR SOME TEMPORARY LOCATION T.

AN EXPRESSION ENCLOSED IN SQUARE BRACKETS PRECEDED BY AN IDENTIFIER IS AN ARRAY REFERENCE. AN EXPRESSION ENCLOSED IN SQUARE BRACKETS BUT NOT PRECEDED BY AN IDENTIFIER IS AN ADDRESS SPECIFED BY INDIRECTION. FOR EXAMPLE, [Y+1] ← 3; SAYS THAT THE CONSTANT 3 IS TO BE STORED IN THE LOCATION WHOSE ADDRESS IS TO BE FOUND BY CALCULATING THE VALUE OF THE EXPRESSION Y+1. IF THE CONTENTS OF Y IS 100 WE STORE 3 IN LOCATION 101.

TO REFER TO REGISTERS OR SENSE SWITCHES IN AN EXPRESSION OR ON THE LEFT OF AN ASSIGNMENT, ENCLOSE THE NAME (AR,BR,XR,OF, SS1,SS2,SS3) IN PARENTHESES.

A FUNCTION CALL IS INDICATED BY A FUNCTION NAME FOLLOWED BY A LIST OF ZERO TO THREE ARGUMENTS ENCLOSED IN PARENTHESES. IF THE FUNCTION HAS NO ARGUMENTS THE PARENTHESES ARE STILL REQUIRED. A FUNCTION RETURNS ITS VALUE IN THE A-REGISTER.

THE VALUE OF A LOGICAL OR RELATIONAL EXPRESSION IS ZERO IF THE EXPRESSION IS FALSE; OTHERWISE IT IS NONZERO.

## BUMP STATEMENT.

THE BUMP STATEMENT ALLOWS THE PROGRAMMER TO TAKE ADVANTAGE OF THE MACHINE INSTRUCTION 'INR' WHICH WILL ADD ONE TO THE CONTENTS OF ANY MEMORY LOCATION.

EXAMPLES:

```
        BUMP I;
        BUMP A,B,C[I-5],[T],"GAMMA+5";
```

THE LIST OF STORAGE LOCATIONS IS SIMILAR TO THAT WHICH MAY APPEAR ON THE LEFT OF AN ASSIGNMENT STATEMENT. THE LOCATIONS ARE INCREMENTED LEFT TO RIGHT ON THE LIST.

NOTE THE EXAMPLE OF A STORAGE LOCATION INDICATED BY SOME DAS ADDRESS EXPRESSION ENCLOSED IN QUOTES. BUMP "GAMMA+5" SAYS TO INCREMENT THE MEMORY LOCATION FIVE WORDS UP FROM THE LOCATION WITH THE NAME GAMMMA.(,INR,GAMMA+5 IN DAS CODE). SO IF THE ADDRESS OF GAMMA IS 349 THEN LOCATION 354 IS INCREMENTED.

## NULL STATEMENT.

THE NULL STATEMENT GENERATES NO MACHINE CODE. IT MAY BE USED AS A DUMMY IN AN IF...THEN...ELSE STATEMENT OR AS SOMETHING TO WHICH A LABEL CAN BE ATTACHED.

EXAMPLES:

```
        LABEL: NULL;
        IF A=5 THEN IF B=3 THEN CALL SUB3 ELSE NULL ELSE CALL SUB5;
```

## GO TO STATEMENT AND LABELS.

A LABEL MAY BE ASSOCIATED WITH A STATEMENT BY PRECEDING THE STATEMENT WITH AN IDENTIFIER AND A COLON. UNCONDITIONAL TRANSFER OF CONTROL IS DONE WITH THE GO TO STATEMENT.

EXAMPLES:

```
        L: GO TO L;
        GOTO LOOP;
        GO TO "FIRST-5";
        LOOP: GO TO 5;
        GOTO [J];
        GOTO X[N];
```

THE GOTO MAY BE WRITTEN AS ONE WORD OR TWO. THE BRANCH MAY BE TO A PARTICULAR LOCATION GIVEN BY A NUMBER, SAY LOCATION 5. THE GO TO MAY ALSO BRANCH INDIRECTLY THROUGH SOME OTHER ADDRESS. THE LAST EXAMPLE WILL CAUSE CONTROL TO BE TRANSFERRED TO THE NTH ELEMENT OF ARRAY X.

## CALL STATEMENT.

A SUBROUTINE IS INVOKED USING THE CALL STATEMENT.

EXAMPLES:
```
        CALL SUB1;
        CALL ABC(X,Y);
        CALL HELP (I);
        CALL CHAR(@J);
        CALL FNX(5,J,K);
        CALL [Q](L);
```

AN ARGUMENT LIST MAY OPTIONALLY FOLLOW THE SUBROUTINE NAME. IT CORRESPONDS IN FORM TO THE ARGUMENT LIST OF A FUNCTION.

WHEN A FUNCTION OR SUBROUTINE IS INVOKED, THE FIRST ARGUMENT IS EVALUATED AND LOADED IN THE A-REGISTER, THE SECOND ARGUMENT IN THE B-REGISTER, AND THE THIRD ARGUMENT IN THE X-REGISTER.

IN THE LAST EXAMPLE ABOVE, THE NAME OF THE SUBROUTINE IS SPECIFIED INDIRECTLY.

TO PASS AN ADDRESS TO A PROCEDURE, OR TO DO SOME CALCULATION ON THE ADDRESS OF A VARIABLE, USE THE VARIABLE NAME PRECEDED BY AN '@' SIGN. FOR EXAMPLE, R←@S; SAYS TO STORE THE ADDRESS OF S IN R.

## RETURN STATEMENT.

AN EXIT FROM A SUBROUTINE, OPTIONALLY RETURNING AN ARGUMENT LIST, IS DONE WITH THE RETURN STATEMENT.

EXAMPLES:
```
        RETURN;
        RETURN (A*6);
        RETURN (X,,Y);
        RETURN FROM LOC ;
        RETURN (X,Y,Z) FROM HEAD;
```

IF AN ARGUMENT LIST APPEARS, THE FIRST ARGUMENT IS EVALUATED AND PUT IN THE A-REGISTER, THE SECOND IN THE B-REGISTER, THE THIRD IN THE X-REGISTER. AN INDIRECT GOTO WITH ARGUMENT PASSING CAN BE ACCOMPLISHED WITH THE RETURN...FROM... FORM. 'RETURN FROM LOC;' IS EQUIVALENT TO 'GO TO [LOC];'.

## STOP STATEMENT.

THE STOP  STATEMENT IS USED TO CAUSE THE MACHINE TO HALT AT RUN TIME.

EXAMPLES:
```
        STOP;
        HALT;
```

## IF STATEMENT.

CONDITIONAL EXECUTION OF A STATEMENT CAN BE ACCOMPLISHED THROUGH USE OF THE IF STATEMENT.

EXAMPLES:

```
        IF A=B THEN H←1 ELSE H←2;
        IF F(N) THEN CALL SUB1(N);
        IF ALPHA THEN BEGIN A←1; GR←-77; K←0; END;
        IF BETA>=7 AND BETA<=76 THEN CALL PRINT(BETA);
        IF (SS1) THEN GOTO TEST1 ELSE IF (SS2) AND (OF) THEN A3←Y;
        IF Z=Y THEN BEGIN Z=Y-1; BUMP Z; END;
```

THE EXPRESSION FOLLOWING THE WORD 'IF' IS EVALUATED TO A BOOLEAN RESULT. IF THE RESULT IS TRUE THE 'THEN' PART IS EXECUTED. IF THE RESULT IS FALSE AND THERE IS NO 'ELSE' PART, THE STATEMENT FOLLOWING THE 'IF' IS EXECUTED. IF THE RESULT OF THE 'IF' EXPRESSION IS FALSE AND THERE IS AN 'ELSE' PART, THAT IS EXECUTED. NOTE THAT STATEMENTS MAY BE GROUPED TOGETHER WITH THE BEGIN...END CONSTRUCT.

'ELSE' CLAUSES IN NESTED 'IF' STATEMENTS ARE ASSOCIATED WITH THE CLOSEST PRECEDING 'IF' THAT HAS NO 'ELSE' ASSOCIATED WITH IT.

## FOR STATEMENT.

A 'FOR' STATEMENT IS USED FOR ITERATION ACROSS A STATEMENT OR BLOCK OF STATEMENTS. THE STORAGE LOCATION INDICATED IN THE ASSIGNMENT PART OF THE STATEMENT IS CALLED THE INDEX VARIABLE. IT IS INITIALIZED, AND THEN THE TEST OF THE 'FOR' STATEMENT IS PERFORMED. IF THE TEST IS AN 'UNTIL' FORM THE VALUE OF THE INDEX VARIABLE IS COMPARED WITH THE 'UNTIL' EXPRESSION. IF THE INDEX VARIABLE IS LESS THAN OR EQUAL TO THE VALUE OF THE EXPRESSION (GREATER THAN OR EQUAL IF THE STEP EXPRESSION IS NEGATIVE), THEN THE 'DO' PART IS EXECUTED. OTHERWISE, CONTROL PASSES TO THE STATEMENT AFTER THE 'FOR' STATEMENT.

IF THE TEST IS A 'WHILE' FORM, THE VALUE OF THE 'WHILE' EXPRESSION IS COMPUTED. IF THE VALUE OF THE EXPRESSION IS TRUE, THE 'DO' PART IS EXECUTED. OTHERWISE, CONTROL PASSES TO THE STATEMENT AFTER THE 'FOR' STATEMENT.

WHENEVER THE 'DO' PART HAS BEEN EXECUTED, THE INDEX VARIABLE IS INCREMENTED BY THE VALUE OF THE 'STEP' EXPRESSION. THE 'STEP' EXPRESSON IS REEVALUATED EACH TIME THROUGH THE LOOP. IF THE 'STEP' EXPRESSION IS MISSING, THEN THE INDEX VARIABLE IS INCREMENTED BY ONE. AFTER THE INDEX VARIABLE IS INCREMENTED, CONTROL PASSES BACK TO THE 'UNTIL' OR 'WHILE' TEST AND THE THE LOOP BEGINS AGAIN.

EXAMPLES:

```
        FOR I←1 UNTIL 10 DO A[I]←I*I;
        FOR I←N STEP -1 UNTIL 1 DO A[1]←A[I]/(I*I);
        FOR J←K-3 STEP M+J WHILE V<R DO V←V+J;
        FOR I←1 WHILE A+B>C DO BEGIN CALL S(I); M[I]←M[I]-1; END;
```

THE FIRST STATEMENT WILL SET A[I]=I*I FOR 1<=I<=10.
THE SECOND IS AN EXAMPLE OF COUNTING THE INDEX VARIABLE DOWN.
THE THIRD USES A 'WHILE' TEST FOR TERMINATION OF THE LOOP. THE LAST HAS A BEGIN BLOCK AS THE 'DO' PART.

## WHILE STATEMENT.

THIS STATEMENT PROVIDES A MEANS OF PERFORMING A STATEMENT AS LONG AS SOME CONDITION IS TRUE.

EXAMPLES:

```
        WHILE A<B DO CALL SUB1(@A,@B);
        WHILE I<=N DO BEGIN A[I]←I; BUMP I; END;
        WHILE X#0 DO X←LINK[X];
```

THE 'WHILE' CONDITION IS TESTED. IF THE RESULT IS TRUE THEN THE 'DO' PART IS EXECUTED. THE TEST AND EXECUTION ARE REPEATED UNTIL THE 'WHILE' EXPRESSION IS FALSE. CONTROL THEN PASSES TO THE NEXT STATEMENT.

## COMPOUND STATEMENT (BLOCK).

A BLOCK IS A GROUP OF STATEMENTS PRECEDED BY THE WORD 'BEGIN' AND FOLLOWED BY THE WORD 'END'. A BLOCK IS USED TO GROUP STATEMENTS LOGICALLY TOGETHER FOR THE 'IF', 'WHILE' AND 'FOR' STATEMENTS. BLOCKS MAY BE NESTED TO ANY LEVEL.

EXAMPLES:

```
        BEGIN A←1; B←2; C←3; END;
        FOR I←1 UNTIL N DO BEGIN A[I]←I; B[I]←0; END;
        IF X=0 THEN BEGIN F←7; GO TO LABEL1; END;
```

## DAS STATEMENT.

A DAS STATEMENT IS A STRING OF CHARACTERS ENCLOSED IN QUOTES ("). THE STRING IS COPIED TO THE OUTPUT FILE WITHOUT QUOTES. THUS ASSEMBLY LANGUAGE (DAS) STATEMENTS MAY BE INSERTED INTO MOL620 PROGRAMS.

EXAMPLES:

```
        ",SEN,0101,*+4";
        "* THIS COMMENT IS PUT IN THE DAS PROGRAM";
        ",EXC,031";
```

## PROCEDURE.

A PROCEDURE IS A CLOSED SUBROUTINE OR FUNCTION. A PROCEDURE MAY NOT CONTAIN ANOTHER PROCEDURE. A PROCEDURE CONSISTS OF THREE PARTS:

(1) HEAD
(2) BODY
(3) CLOSE.

THE PROCEDURE HEAD CONSISTS OF THE WORD 'PROC' (OR 'PROCEDURE') FOLLOWED BY AN IDENTIFIER WHICH SERVES AS THE PROCEDURE NAME, FOLLOWED BY AN OPTIONAL LIST OF 0 TO 3 IDENTIFIERS ENCLOSED IN PARENTHESES.

EXAMPLES OF PROCEDURE HEADS:

```
        PROC ALPHA(X);
        PROCEDURE F(Q1,R,V);
        PROC SUB1;
        PROCEDURE SUB1()
        PROC G3(X,,Z);
```

IF A PARAMETER LIST APPEARS IN THE PROCEDURE HEADING, THEN ON ENTRY TO THE PROCEDURE THE A-REGISTER IS STORED IN THE LOCATION SPECIFIED BY THE FIRST PARAMETER, THE B-REGISTER IN THE SECOND, THE X-REGISTER IN THE THIRD. THE PARAMETERS ARE NOT DUMMIES IN THE SENSE THAT THEY ARE NOT LOCAL TO THE PROCEDURE BUT HAVE SCOPE OVER THE ENTIRE DAS PROGRAM GENERATED.

THE BODY OF A PROCEDURE IS EMPTY OR CONSISTS OF A SEQUENCE OF STATEMENTS, COMMENTS, AND DECLARATIONS.

A PROCEDURE IS CLOSED BY THE WORD 'ENDP;'. THIS HAS THE SAME EFFECT AS THE RETURN STATEMENT WITH NO ARGUMENTS.

EXAMPLES:

```
        PROCEDURE ADD (ARG1,ARG2);
        RETURN (ARG1+ARG2);
        DECLARE ARG1,ARG2;
        ENDP;

        PROCEDURE LINEARSEARCH (FIRST,N,VALUE);
        %THE ARGUMENTS ARE:
         FIRST = ADDRESS OF VECTOR TO BE SEARCHED
         N     = SIZE OF ARRAY (1 TO N)
         VALUE = VALUE TO BE FOUND.
         WE USE A LINEAR SEARCH TECHINIQUE TO RETURN
         THE INDEX OF THE VECTOR ELEMENT WHICH
         CONTAINS THE VALUE. (ASSUME IT IS THERE)%
        LAST←FIRST+N-1; %ADDR OF LAST WORD IN VECTOR%
        FOR I←FIRST UNTIL LAST DO
         IF [I]=VALUE THEN RETURN (I-FIRST+1);
        DECLARE FIRST,LAST,I,N,VALUE;
        ENDP;
```

A PROCEDURE SHOULD ONLY BE ENTERED VIA A FUNCTION CALL OR SUBROUTINE CALL, AND NOT BY FALLING INTO IT FROM PRECEDING CODE.

DECLARATIONS.

THE USER MUST DECLARE STORAGE FOR VARIABLES WHICH ARE SCALARS OR ONE DIMENSIONAL VECTORS. AN INITIAL VALUE MAY ALSO BE SPECIFIED FOR THESE VARIABLES. DECLARATIONS ARE GLOBAL TO THE WHOLE MOL620 PROGRAM.

EXAMPLES:

```
        DECLARE A,B,C,D=1,E[10],F[-5:6];
        DECL G[1:5]="1,2,3,4,5",TEXT="'ABCDEFGHIJK'";
```

THIS GENERATES THE FOLLOWING CODE:

```
         ,JMP,$4
        A,DATA,0
        B,DATA,0
        C,DATA,0
        D,DATA,1
        E,DATA,0
         ,ORG,E+10
        F,EQU,*+5
         ,ORG,F+6+1
        $4,EQU,*
         ,JMP,$5
        G,EQU,*-1
         ,DATA,1,2,3,4,5
         ,ORG,G+5+1
        TEXT,DATA,'ABCDEFGHIJK'
        $5,EQU,*
```

NOTE THE UNCONDITIONAL JUMPS AROUND THE DECLARATIONS. EXECUTION OF DECLARATIONS CANNOT OCCUR BY FALLING INTO THEM FROM PRECEDING CODE.

THE USER MAY PROVIDE FOR THE INDEXING OF VECTORS USING A VALUE OF AN INDEX FROM M TO N BY DECLARING THE VECTOR (SAY V) AS V[M:N] WHERE M AND N HAVE VALUES AT ASSEMBLY TIME. SINCE MOL620 GENERATES CONDITIONAL ASSEMBLY STATEMENTS FOR INDEXED INSTRUCTIONS, ARRAY NAMES MUST BE DECLARED BEFORE THEY ARE USED IF THE ASSEMBLY TIME VALUE OF THE NAME IS GREATER THAN 511 (777 BASE 8). OTHERWISE, PHASE ERRORS WILL RESULT.

TO DECLARE SYMBOLS TO HAVE A CERTAIN VALUE AT ASSEMBLY TIME WE HAVE ANOTHER FORM OF THE DECLARATION:

EXAMPLES:

```
        SET A=0, B=7, D=6;
        EQU E=5,F=01777;
```

GENERATES THE DAS CODE

```
        A ,SET, 0
        B ,SET, 7
        D ,SET, 6
        E ,EQU, 5
        F ,EQU, 01777
```

COMMENTS.

COMMENTS ARE OF TWO TYPES:

(1) STRINGS ENCLOSED IN PERCENT SIGNS (%). THESE ARE MERELY ANNOTATIONS AND DO NOT APPEAR IN THE OUTPUT OF MOL620.

(2) STRINGS ENCLOSED IN QUOTES(") AND FOLLOWED BY A SEMI-COLON. THE QUOTES ARE REMOVED AND THE STRING IS COPIED TO THE OUTPUT FILE. (SEE DAS STATEMENT).

PROGRAM ORIGIN.

IN ORDER TO GIVE THE DAS PROGRAM AN ORIGIN ADDRESS, THE PROGRAM MAY CONTAIN A CONSTRUCT TO INDICATE THE ORG LOCATION.
THE ORG APPEARS BEFORE ANY EXECUTABLE STATEMENTS OF THE PROGRAM.

EXAMPLES:

```
        P3 ORG 013000;
        MAIN ORG 5000;
```

IF THE ORG ADDRESS IS AN ADDRESS EXPRESSION INVOLVING VARIABLES, THESE VARIABLES MUST BE DECLARED PREVIOUSLY WITH A SET OR EQU.

EXAMPLE:

```
        SET B=013000,C=0257;
        ALPHA ORG B+C;
```

IN THIS CASE ALPHA IS SET AT LOCATION 013257 AT ASSEMBLY TIME.

PROGRAM.

A PROGRAM IS A SET OF COMMENTS, DECLARATIONS AND PROCEDURES OPTIONALLY PRECEDED BY THE PROGRAM ORIGIN.

## MOL620 COMPILER AND LANGUAGE. HINTS, PITFALLS, AND MISCELLANY.

### SYMBOLS.

MOL620 WILL ACCEPT AN IDENTIFIER (IDEN) OF INDEFINITE LENGTH, HOWEVER, DAS WILL NOT DIFFERENTIATE BETWEEN SYMBOLS IF THE FIRST FOUR CHARACTERS ARE IDENTICAL.

DO NOT USE SYMBOLS BEGINNING WITH A DOLLAR SIGN ($). MOL620 RESERVES SUCH SYMBOLS FOR GENERATING LABELS.

A PERCENT SIGN (%) MAY ONLY BE USED AS A COMMENT DELIMITER.

A ZERO IN FRONT OF A NUMBER INDICATES THE NUMBER IS EXPRESSED IN BASE 8.

A CARRIAGE RETURN MAY NOT APPEAR IN A STRING OF CHARACTERS DELIMITED BY DOUBLE QUOTES (").

### OPERATORS AND EXPRESSION EVALUATION.

THE HIERARCHY OF THE PRIMITIVE OPERATORS OF MOL620 IS:

```
        UNARY +, UNARY -           (HIGHEST BINDING)
        "IDEN"   (.E.G. "ASRA", "LLRL", "MUL", I.E.
                          ANY DAS OPCODE IN QUOTES)
        BAND
        BOR, BXR
        *, /
        +, -
        >, >=, =, <=, <, #
        NOT
        AND
        OR, XOR                    (LOWEST BINDING)
```

BAND, BOR, AND BXR PERFORM THE BITWISE AND, OR, AND EXCLUSIVE OR OF THE TWO OPERANDS.

THE ONE'S COMPLEMENT OF A VALUE V CAN BE COMPUTED AS V BXR -1.

THE LOGICAL AND RELATIONAL OPERATORS RETURN A VALUE OF TRUE (NON-ZERO) OR FALSE (ZERO).

THE OVERFLOW INDICATION IN MULTIPLICATION IS NOT SIGNALLED.

A VALUE V MAY BE LOADED INTO THE A-REGISTER AND SHIFTED RIGHT 7 PLACES BY V "LSRA" 7. (IN DAS CODE: ,LDA,V     ,LSRA,7)

TWO OPERATORS MAY NOT APPEAR TOGETHER. WRITE P*-Q AS P*(-Q).

IN MOL620, MULTIPLICATION BY 2,4,8,16, AND 256 IS DONE BY THE APPROPRIATE ARITHMETIC SHIFT. NO OVERFLOW IS INDICATED.

DIVISION BY 2 IS DONE BY AN ARITHMETIC SHIFT RIGHT.

DIVISION OF NEGATIVE NUMBERS MAY BE IN ERROR DUE TO HARDWARE DIVIDE PROBLEMS ON THE 620/I.

THE VALUE OF AN EXPRESSION OF AND'S AND OR'S IS COMPUTED ONLY SO FAR AS NECESSARY TO ASSURE A CORRECT RESULT. FOR EXAMPLE, THE EXPRESSION A AND B AND C IS EVALUATED LEFT TO RIGHT. IF A IS FALSE, THEN THE EXPRESSION IS FALSE AND THE OTHER OPERANDS ARE NOT EVALUATED. IF A IS TRUE, THEN B IS EVALUATED, AND SO ON.

THE ORDER OF EVALUATION OF ARITHMETIC EXPRESSIONS IS NORMALLY LEFT TO RIGHT IF THE EXPRESSION IS NOT PARENTHESIZED. HOWEVER, THE ORDER OF OPERATIONS MAY BE REARRANGED BY THE COMPILER FOR THE SAKE OF EFFICIENCY. CARE MUST BE EXERCISED WHEN TAKING ADVANTAGE OF THE SIDE EFFECTS OF ANY EXPRESSION EVALUATION OR WHEN YOU WANT NORMALLY COMMUTATIVE OPERATIONS TO BE DONE IN A SPECIFIC ORDER. IT MAY BE NECESSARY TO CHECK THE OUTPUT OF THE COMPILER TO MAKE CERTAIN THAT A SPECIFIC ORDER OF EVALUATION WAS FOLLOWED.

WHENEVER AN EXPRESSION APPEARS IN A PROGRAM , THE CODE WHICH IS GENERATED WILL LOAD THE VALUE OF THAT EXPRESSION INTO THE A-REGISTER OF THE MACHINE. THEREFORE ALL EXPRESSIONS MUST EVALUATE TO A 16 BIT VALUE. IN THE CASE OF MULTIPLICATION, WHICH HAS A TWO WORD RESULT WHEN DONE BY THE HARDWARE, THE A-REGISTER CONTAINS THE PRODUCT MOD 2**15. FOR DIVISION, THE QUOTIENT IS DEVELOPED IN THE A-REGISTER AND THE INTEGER REMAINDER IS IN THE B-REGISTER.

FINALLY, IF IT IS UNCLEAR WHAT CODE A CERTAIN SYNTACTIC CONSTRUCT WILL GENERATE, RUN THE COMPILER AND SEE. (INPUT DEVICE=TTY:, OUTPUT DEVICE=TTY:).

## MINIMUM 620/I CONFIGURATION.

TO SUCCESSFULLY RUN MOL620 PROGRAMS THE 620/I MUST BE EQUIPPED WITH THE EXTENDED ADDRESSING--HARDWARE MULTIPLY/DIVIDE OPTION. THE COMPILER COULD BE ALTERED IN ABOUT ONE MAN HOUR TO REMOVE THIS RESTRICTION. LOW CORE INDIRECT ADDRESSING AND A PROGRAM CALL TO A MULTIPLY/DIVIDE ROUTINE WOULD THEN APPEAR IN THE OUTPUT OF MOL620.

## HOW TO COPE WITH THE MOL620 COMPILER.

MOL620 EXPECTS AN INPUT FILE WRITTEN IN MOL620 LANGUAGE AND OUTPUTS A FILE WHICH IS THE DAS ASSEMBLY LANGUAGE TRANSLATION OF THE INPUT.

IF A SYNTAX ERROR OCCURS, MOL620 WILL LIST THE LINE IN ERROR AND STOP. FIX THE ERROR AND COMPILE IT AGAIN. MOL620 ONLY DETECTS ONE ERROR AT A TIME. THE USE OF A LINE-ORIENTED TEXT EDITOR (SUCH AS QED OR TECO) IS HELPFUL AS THE STATEMENTS IN ERROR ARE INDICATED BY THEIR POSITION FROM THE START OF THE FILE. THE ERROR MESSAGE NUMBERS GIVEN BY MOL620 ARE MEANINGLESS.

SOME (NOT MANY) ERRORS CAUSE AN ILLEGAL TERMINATION OF THE COMPILER WITH NO INDICATION OF WHAT CAUSED THE ERROR. EXAMINE THE PARTIALLY BUILT OUTPUT FILE TO SEE WHERE THE COMPILER WAS WHEN IT BLEW UP OR RUN IT AGAIN WITH THE OUTPUT TO THE TELETYPE.

MOL620 RUNS IN ABOUT 15K UNDER THE STANDARD DEC PDP-10 MONITOR AND COMPILED THE 100 LINE EXAMPLE IN THE APPENDIX IN 4 SECONDS OF CPU TIME.

## RUNNING THE MOL620 COMPILER ON THE UCI PDP-10 SYSTEM.

(1) PREPARE THE SOURCE PROGRAM AND STORE IT ON DSK:FILE.MOL.

(2) TYPE THE FOLLOWING:

```
.RUN DSK:MOL620[XXX,YYY]
*INPUT.DAS←FILE.MOL
```

(3) THE COMPILER SHOULD RUN AND OUTPUT THE DAS CODE TO INPUT.DAS.

(4) RUN PIP TO GET A PAPER TAPE COPY TO ASSEMBLE ON THE VARIAN:

```
.AS PTP
.R PIP
*PTP:←INPUT.DAS
*↑C
.DEAS PTP
```

OR

(5) ASSEMBLE YOUR DAS PROGRAM ON THE PDP-10 AND PUNCH THE BINARY TAPE:

```
.R PIP
*/X←INPUT.DAS/A
*↑C
.RUN DSK:DAS10[XXX,YYY]
*$  (ESCAPE OR ALTMODE)
.AS PTP
.R PIP
*PTP:←BINARY.DAS/I
*↑C
.DEAS PTP
```

THE DAS LISTING IS IN FILE INPUT.LST. THE BINARY OUTPUT IS IN FILE BINARY.DAS. XXX,YYY IN THE ABOVE PROTOCOL IS THE USER NUMBER OF THE ACCOUNT WHERE THE MOL620 COMPILER RESIDES.

## HISTORY.

## DESIGN CRITERIA.

THE PURPOSE OF THE MOL620 PROJECT, WHICH BEGAN IN 1968, WAS TO DESIGN AND IMPLEMENT A TRANSLATOR FOR THE DEPARTMENT OF INFORMATION AND COMPUTER SCIENCE VARIAN DATA 620/I, AN 8K, 16 BIT/WORD MINI-COMPUTER. THE ONLY TRANSLATOR THEN AVAILABLE WAS A SIMPLE TWO-PASS ASSEMBLER.

IT WAS ENVISIONED THAT BOTH FACULTY AND STUDENTS WOULD BE INTERESTED IN USING THE 620/I SO IT WAS NECESSARY TO DESIGN A LANGUAGE AND IMPLEMENTATION THAT WOULD BE EASY FOR INEXPERIENCED STUDENTS TO USE, YET VERSATILE ENOUGH TO BE USED IN SIGNIFICANT RESEARCH. GIVEN THE EXPECTED CLASS OF USERS, THE TYPE OF WORK THEY WOULD BE INTERESTED IN DOING, AND THE NATURE OF THE COMPUTER AVAILABLE, THE FOLLOWING CRITERIA FOR THE LANGUAGE DESIGN SEEMED APPROPRIATE:

(1) THE LANGUAGE SHOULD BE EASY TO LEARN AND MASTER. THIS WAS ESPECIALLY IMPORTANT SINCE THE MAJORITY OF USERS WERE EXPECTED TO BE STUDENTS WHO WOULD PROBABLY DEVOTE NO MORE THAN A WEEK OR TWO TO LEARNING THE LANGUAGE. SYNTACTICALLY CORRECT CONSTRUCTS SHOULD HAVE OBVIOUS MEANINGS; CONSTRUCTIONS WHICH SEEM NATURAL AND SEEM TO HAVE STRAIGHTFORWARD MEANINGS SHOULD NOT BE EXCLUDED.

(2) THE SYNTAX SHOULD BE AS SIMPLE AS POSSIBLE. A WELL-CONSTRUCTED SYNTAX WHICH CLEARLY REFLECTS THE STRUCTURE OF THE LANGUAGE IS USEFUL FOR CHECKING THE CORRECTNESS OF STATEMENTS BEFORE ATTEMPTING TO TRANSLATE THEM AND FOR PINNING DOWN SYNTACTIC ERRORS DETECTED BY A TRANSLATOR.

(3) THE LANGUAGE SHOULD BE EASY TO READ YET SHOULD BE ORIENTED TOWARD SIMPLE TELETYPE KEYBOARD ENCODINGS. SOMEONE WITH LITTLE OR NO EXPOSURE TO THE LANGUAGE SHOULD BE ABLE TO PICK UP A LISTING AND MAKE A REASONABLE GUESS ABOUT WHAT THE PROGRAM DOES. THE NEW USER SHOULD NOT BE FRIGHTENED BY AN AWESOME DISPLAY OF ASTERISKS, SHARP SIGNS, BRACKETS, DOLLAR SIGNS, QUOTES AND OTHER SYMBOLS ONLY OCCASIONALLY INTERSPERSED WITH RECOGNIZABLE WORDS. SIMPLICITY OF ENCODINGS IS IMPORTANT SINCE THE TELETYPE IS SLOW AND AWKWARD TO USE, BUT READABILITY OF THE LANGUAGE IS MORE IMPORTANT.

(4) ALGORITHMS SHOULD HAVE CLEAR ENCODINGS IN THE LANGUAGE; FLOW OF CONTROL SHOULD ALSO BE CLEAR. PROGRAMMING IN ASSEMBLY LANGUAGE IS TEDIOUS AND THE RESULTING PROGRAM IS OFTEN DIFFICULT TO FOLLOW, EVEN IF EXTENSIVELY COMMENTED. A LANGUAGE MORE STRUCTURED THAN ASSEMBLY LANGUAGE, E.G. WITH AN ALGOL-LIKE STRUCTURE, IS MUCH EASIER TO FOLLOW. IN ADDITION, A PROGRAM IN A STRUCTURED LANGUAGE WILL BE PARTIALLY SELF-DOCUMENTING, I.E. A PROGRAM WILL NOT SUFFER AS BADLY IF THERE IS LITTLE OR NO DOCUMENTATION. IT IS OFTEN EASIER IN A STRUCTURED PROGRAMMING

LANGUAGE THAN IN ASSEMBLY LANGUAGE FOR THE PROGRAMMER TO DETECT AND CORRECT HIS ERRORS.

(5) THE LANGUAGE SHOULD ALSO REFLECT THE STRUCTURE OF THE TARGET COMPUTER SO THAT, WHEN NECESSARY, OPTIMAL USE CAN BE MADE OF THE LIMITED AMOUNT OF MEMORY AVAILABLE.

ONE TYPE OF LANGUAGE WHICH MET THESE CRITERIA IS A MACHINE ORIENTED LANGUAGE. SOME OF THE FEATURES OF SUCH A LANGUAGE ARE:

(1) THE LANGUAGE IS MORE STRUCTURED THAN ASSEMBLY LANGUAGE.
(2) ALL MEMORY LOCATIONS AND MACHINE REGISTERS ARE DIRECTLY ACCESSIBLE.
(3) THE LANGUAGE ALLOWS THE SPECIFICATION OF GENERAL COMPUTATIONS AND ASSIGNMENTS AT A MACHINE-INDEPENDENT LEVEL, WITH THE DETAILS LEFT TO THE LANGUAGE TRANSLATOR.
(4) THE LANGUAGE ALLOWS THE ENTRY OF ASSEMBLY LANGUAGE CODE DIRECTLY.
(5) THE LANGUAGE DOES NOT MASK ANY OF THE CAPABILITIES OF ASSEMBLY LANGUAGE, BUT MAKES EASIER AND MORE EXPLICIT THE EXPRESSION OF THOSE CAPABILITIES USED MOST FREQUENTLY.

MOL620 WAS DESIGNED TO INCLUDE THE CAPABILITIES COMMON TO MOST MACHINE ORIENTED LANGUAGES AND TO MEET THE CRITERIA DESCRIBED ABOVE. THIS ORIGINAL DESIGN HAS BEEN FOLLOWED THROUGH TWO IMPLEMENTATIONS.

## COMPARISON WITH OTHER MACHINE ORIENTED LANGUAGES.

SEVERAL MACHINE ORIENTED LANGUAGES (MOLS), IN PARTICULAR MOL-32 FOR THE Q-32 COMPUTER AT SYSTEM DEVELOPMENT CORPORATION, MOL940 FOR THE SDS 940 COMPUTER AT STANFORD RESEARCH INSTITUTE, AND PL360 FOR THE IBM 360/67 COMPUTER AT STANFORD UNIVERSITY, WERE EXAMINED AND THE LIMITATIONS AND PECULIARITIES OF THE DATA 620/I WERE CONSIDERED. A LIST OF THE FEATURES TO BE INCLUDED IN A MOL FOR THE 620/I (HEREAFTER CALLED MOL620) WAS THEN PRODUCED. THIS LIST WAS THEN PRUNED TO WHAT APPEARED TO BE NECESSARY FOR A USABLE FIRST PASS. WHAT REMAINED INCLUDED DECLARATION AND INITIALIZATON OF VARIABLES, COMMENTS, AND CONDITIONAL, LOOP, ASSIGNMENT, NULL, BLOCK (BEGIN-END), SUBROUTINE CALL, SUBROUTINE RETURN, INCREMENT, AND ASSEMBLY LANGUAGE STATEMENTS.

SEVERAL FORMS WERE PROPOSED FOR EACH OF THE LANGUAGE CONSTRUCTS AND ALTERNATIVES WERE DISCARDED PRIMARILY ON AESTHETIC GROUNDS. ALTHOUGH THE TYPES OF CONSTRUCTS REMAINING REFLECT MACHINE STRUCTURE TO SOME DEGREE, THE FORMS OF THE CONSTRUCTS DO NOT. IN MANY CASES IMPLEMENTATION COULD HAVE BEEN SIMPLIFIED IF SLIGHT BUT OBSCURE (FROM THE USER'S POINT OF VIEW) AND POSSIBLY ANNOYING CHANGES IN THE FORM OF CONSTRUCTS HAD BEEN MADE; THIS APPROACH WAS REJECTED. OTHER MOLS, E.G. MOL940, REQUIRE SOMEWHAT AWKWARD CONSTRUCTS. FOR EXAMPLE, IN MOL940 A LABEL MUST BE ENCLOSED IN PARENTHESES AND FOLLOWED BY A COLON.

MOL620 RESEMBLES MOL940 MORE THAN PL360 OR MOL-32 FOR TWO REASONS:

(1) THE VARIAN DATA 620/I HAS SOME RESEMBLANCE TO THE SDS 940. BOTH MACHINES HAVE THREE USER ACCESSIBLE REGISTERS AND HAVE SIMILAR ORDER CODES.

(2) MOL940 COMBINES SOME OF THE BEST FEATURES OF EARLIER MACHINE ORIENTED LANGUAGES. MUCH OF THE DESIGN SEEMED WORTH FOLLOWING.

MOL620 DIFFERS FROM MOL940 IN EXTENT AND INTENT. MOL940 IS A MORE EXTENSIVE LANGUAGE THAN MOL620. MOL940 INCLUDES SEVERAL CONSTRUCTS WHICH ALLOW THE USER TO AFFECT THE OPTIMALITY OF THE OUTPUT CODE. IT ALSO REFLECTS THE POWER OF THE 940 ASSEMBLER. THE MOL940 LANGUAGE AND COMPILER ARE ORIENTED TOWARD PRO-DUCING EFFICIENT AND CONCISE CODE ON A DISPLAY-ORIENTED TIME-SHARING COMPUTER. IN THE MOL620 LANGUAGE AND COMPILER, THE CHIEF CONCERNS ARE EASE OF USE AND EASE OF MODIFICATION. THE PACKAGE IS NOT INTENDED TO BE STRICTLY DIRECTED TOWARD EXPERIENCED PROGRAMMERS. A STUDENT SHOULD BE ABLE TO USE THE COMPILER FOR CLASS PROJECTS. THUS THE LANGUAGE AND LANGUAGE COMPILER ARE VIEWED BOTH AS A VEHICLE FOR DOING USEFUL WORK AND AS A LEARNING TOOL.

## BIBLIOGRAPHY.

(1) BOOK, E. AND D.V. SCHORRE, "A HIGHER-LEVEL MACHINE-ORIENTED LANGUAGE AS AN ALTERNATIVE TO ASSEMBLY LANGUAGE," TECH MEMO 3086/001/00, SYSTEM DEVELOPMENT CORPORATION.

(2) CARR, C. STEPHEN ET.AL., "THE TREE-META COMPILER-COMPILER SYSTEM," UNIVERSITY OF UTAH.

(3) HAY, R.E. AND J.F. RULIFSON, "MOL940: PRELIMINARY SPECIFICATION FOR AN ALGOL-LIKE MACHINE-ORIENTED LANGUAGE FOR THE SDS 940," INTERIM TECHNICAL REPORT 2, SRI PROJECT 5890, STANFORD RESEARCH INSTITUTE.

(4) RULIFSON, J.F., "A TREE META FOR THE XDS 940," AUGMENTATION RESEARCH CENTER, STANFORD RESEARCH INSTITUTE, MENLO PARK, CALIFORNIA, APRIL 1968.

(5) "VARIAN DATA 620/I COMPUTER MANUAL," VARIAN DATA MACHINES.

(6) WIRTH, N., "PL360, A PROGRAMMING LANGUAGE FOR THE 360 COMPUTERS," JOURNAL OF THE ACM, JANUARY 1968.

## APPENDIX I--SAMPLE MOL620 PROGRAM

```
SPACE ORG 0120; %DYNAMIC STORAGE ALLOC ROUTINES%
%FIRST FIT METHOD; BLOCKS KEPT ON ONE LIST IN
 DESCENDING ORDER BY BLOCK ADDRESS. SECOND TO LAST
 WORD IN BLOCK IS BLOCK SIZE, LAST WORD IS ADDRESS OF
 NEXT BLOCK.
 IF BLOCKS OF SIZE ONE COME UP, THEY ARE PUT ON
 THE SINGLES LIST.%

EQU LINK=1,SIZE=0,ERR=012,STATS=013,EMPTY=016;
",ORG,030";
",DATA,CREATEPOOL,BFREE,LFREE,RES,SFREE";
",ORG,SPACE";

%FREE IS NUMBER OF FREE WORDS AVAILABLE IN THE SYSTEM%

PROC CREATEPOOL(X,N); %SET UP FREE STORAGE AREA%
%X POINTS TO BASE OF FREE STORAGE
 N IS NUMBER OF WORDS UP FROM THAT ADDRESS TO BE
   AVAILABLE%
STATS←@FREE;
FREE,EMPTY,SINGLES←0;
"HEAD+LINK",Q←@NIL;
CALL BFREE(X,N); %PUT BLOCK ON LIST%
ENDP;

PROC BFREE(X,N); %FREE A BLOCK OF N WORDS%
%X POINTS TO LOW ORDER ADDRESS OF BLOCK%
IF X=0 OR N<=0 THEN RETURN;
X←X+N-2; %MAKE X POINT TO HIGH ORDER END OF BLOCK%
FREE←FREE+N;
IF Q>X THEN C←Q ELSE C←@HEAD;
B←LINK[C];
WHILE B>X DO %SEARCH FOR LOWER ADDRESS THAN X%
 BEGIN C←B; B←LINK[B]; END;
LINK[X]←B; SIZE[X]←N; LINK[C]←X; %PUT BLOCK IN%
Q←X; %REMEMBER WHERE WE LAST PUT IN A BLOCK, IT MIGHT
      SAVE US SOME TIME ON NEXT BFREE CALL%
ENDP;

PROC LFREE(LF); %FREE LIST LF OF 2 WORD BLOCKS%
WHILE LF#0 DO
 BEGIN T←LINK[LF]; CALL BFREE(LF,2); LF←T; END;
ENDP;

PROC SFREE(SF); %FREE A STRING IN NORMAL FORM%
N←[SF]/2; CALL BFREE(SF,N+(BR)+1);
ENDP;
```

```
PROC RES(N); %RESERVE A BLOCK OF N WORDS%
IF N<=0 THEN RETURN;
Q←@NIL; %PLAY IT SAFE. WE MIGHT BE GIVING UP BLOCK Q POINTS TO%
COMBINED←0;
IF FREE-N<50 THEN EMPTY←1;
RESA:
S←@HEAD; R←LINK[S];
WHILE SIZE[R]<N DO
 BEGIN S←R; R←LINK[R]; END; %FIND FIRST FIT%
IF R#@NIL THEN
 BEGIN
  R1←SIZE[R]-N; U←R-N+1;
  IF R1=0 THEN LINK[S]←LINK[R];  %PERFECT FIT, UNLINK BLOCK%
  ELSE
   IF R1=1 THEN  %PUT LAST WORD ON SINGLES, UNLINK BLOCK%
    BEGIN [U]←SINGLES; SINGLES←U; LINK[S]←LINK[R]; FREE←FREE-1; END;
    ELSE  %WE ONLY NEED PART OF BLOCK%
     BEGIN T,LINK[S]←R-N; SIZE[T]←R1; LINK[T]←LINK[R]; END;
  %ZERO OUT BLOCK TO BE RETURNED%
  BUMP R,U;
  FOR I←U UNTIL R DO [I]←0;
  FREE←FREE-N;
  RETURN(U); %ADDRESS OF BLOCK%
 END;
%OTHERWISE WE TRY TO COMBINE BLOCKS TOGETHER%
IF COMBINED %ALREADY TRIED% THEN CALL [ERR](4);
 P←@HEAD; R←LINK[P]; L←LINK[R];
WHILE L#@NIL DO
BEGIN
 S←SIZE[R];
 IF R-S=L THEN
  BEGIN SIZE[R]←S+SIZE[L]; LINK[R]←LINK[L]; END;
 ELSE
  BEGIN P←R; R←L; END;
 L←LINK[L];
END;
COMBINED←1;
GOTO RESA; %TRY TO FIND A BLOCK BIG ENOUGH NOW%
ENDP;

DECL X,N,HEAD[2],C,B,FREE,COMBINED,NIL[2]="077777,NIL",
     S,R,R1,I,SINGLES,LF,SF,L,T,P,Q,U;
DONE.
```

## APPENDIX I--OUTPUT OF THE COMPILER FOR SAMPLE PROGRAM

```
* COMPILED BY MOL620  VERSION 2.15  (17-SEPT-71)
,SMRY,
SPACE,ORG,0120
$,EQU,*
LINK,EQU,1
SIZE,EQU,0
ERR,EQU,012
STATS,EQU,013
EMPTY,EQU,016
,ORG,030
,DATA,CREATEPOOL,BFREE,LFREE,RES,SFREE
,ORG,SPACE
$PO,EQU,*
CREATEPOOL,NOP,       <<<<<PROCEDURE CREATEPOOL>>>>>
,STA,X
,STB,N
,LDAI,FREE
,STA,STATS
,TZA,
,STA,SINGLES
,STA,EMPTY
,STA,FREE
,LDAI,NIL
,STA,Q
,STA,HEAD+LINK
,LDA,X
,LDB,N
,JMPM,BFREE
,JMP*,$PO
$TO,EQU,*-1
,BSS,0
$P1,EQU,*
BFREE,NOP,       <<<<<PROCEDURE BFREE>>>>>
,STA,X
,STB,N
,LDA,X
,JAZ,*+3
,DECR,1
,IAR,
,JAZ,*+4
,JMP,$1
,LDA,N
,DAR,
,LSRA,15
$1,EQU,*
,JAZ,$2
,JMP*,$P1
$2,EQU,*
,LDA,X
,ADD,N
,SUBI,2
,STA,X
,LDA,FREE
,ADD,N
```

```
,STA,FREE
,LDA,Q
,SUB,X
,DAR,
,JAN,$3
,LDA,Q
,STA,C
,JMP,$4
$3,EQU,*
,LDAI,HEAD
,STA,C
$4,EQU,*
,LDX,C
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,B
$5,EQU,*
,LDA,B
,SUB,X
,DAR,
,JAN,$6
,LDA,B
,STA,C
,LDX,B
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,B
,JMP,$5
$6,EQU,*
,LDA,B
,LDX,X
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
,LDA,N
,LDX,X
,IFT,SIZE,512,512
,STA,SIZE,1
,IFT,511,SIZE,SIZE
,STAE,SIZE,1
,LDA,X
,LDX,C
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
,LDA,X
,STA,Q
,JMP*,$P1
$T1,EQU,*-1
,BSS,0
```

```
$P2,EQU,*
LFREE,NOP,      <<<<<PROCEDURE LFREE>>>>>
,STA,LF
$7,EQU,*
,LDA,LF
,JAZ,$8
,LDX,LF
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,T
,LDA,LF
,LDBI,2
,JMPM,BFREE
,LDA,T
,STA,LF
,JMP,$7
$8,EQU,*
,JMP*,$P2
$T2,EQU,*-1
,BSS,0
$P3,EQU,*
SFREE,NOP,      <<<<<PROCEDURE SFREE>>>>>
,STA,SF
,LDAE,(SF)*
,TAB,
,LASR,1
,ASRB,14
,STA,N
,TBA,
,ADD,N
,IAR,
,TAB,
,LDA,SF
,JMPM,BFREE
,JMP*,$P3
$T3,EQU,*-1
,BSS,0
$P4,EQU,*
RES,NOP,      <<<<<PROCEDURE RES>>>>>
,STA,N
,LDA,N
,DAR,
,JAP,$9
,JMP*,$P4
$9,EQU,*
,LDAI,NIL
,STA,Q
,TZA,
,STA,COMBINED
,LDA,FREE
,SUB,N
,SUBI,50
,JAP,$10
,INCR,1
,STA,EMPTY
$10,EQU,*
RESA,EQU,*
,LDAI,HEAD
,STA,S
```

```
,LDX,S
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,R
$11,EQU,*
,LDX,R
,IFT,SIZE,512,512
,LDA,SIZE,1
,IFT,511,SIZE,SIZE
,LDAE,SIZE,1
,SUB,N
,JAP,$12
,LDA,R
,STA,S
,LDX,R
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,R
,JMP,$11
$12,EQU,*
,LDA,R
,SUBI,NIL
,JAZ,$13
,LDX,R
,IFT,SIZE,512,512
,LDA,SIZE,1
,IFT,511,SIZE,SIZE
,LDAE,SIZE,1
,SUB,N
,STA,R1
,LDA,R
,SUB,N
,IAR,
,STA,U
,LDA,R1
,JAZ,*+4
,JMP,$14
,LDX,R
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,LDX,S
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
,JMP,$15
$14,EQU,*
,LDA,R1
,SUBI,1
,JAZ,*+4
,JMP,$16
,LDA,SINGLES
,STAE,(U)*
,LDA,U
,STA,SINGLES
,LDX,R
```

```
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,LDX,S
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
,LDA,FREE
,DAR,
,STA,FREE
,JMP,$17
$16,EQU,*
,LDA,R
,SUB,N
,LDX,S
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
,STA,T
,LDA,R1
,LDX,T
,IFT,SIZE,512,512
,STA,SIZE,1
,IFT,511,SIZE,SIZE
,STAE,SIZE,1
,LDX,R
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,LDX,T
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
$17,EQU,*
$15,EQU,*
,INR,R
,INR,U
,LDA,U
,STA,I
$18,EQU,*
,LDA,I
,SUB,R
,DAR,
,LSRA,15
,JAZ,$19
,TZA,
,STAE,(I)*
,INR,I
,JMP,$18
$19,EQU,*
,LDA,FREE
,SUB,N
,STA,FREE
,LDA,U
,JMP*,$P4
$13,EQU,*
,LDA,COMBINED
```

```
,JAZ,$20
,LDAI,4
,JMPM,(ERR)*
$20,EQU,*
,LDAI,HEAD
,STA,P
,LDX,P
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,R
,LDX,R
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,L
$21,EQU,*
,LDA,L
,SUBI,NIL
,JAZ,$22
,LDX,R
,IFT,SIZE,512,512
,LDA,SIZE,1
,IFT,511,SIZE,SIZE
,LDAE,SIZE,1
,STA,S
,LDA,R
,SUB,S
,SUB,L
,JAZ,*+4
,JMP,$23
,LDX,L
,IFT,SIZE,512,512
,LDA,SIZE,1
,IFT,511,SIZE,SIZE
,LDAE,SIZE,1
,ADD,S
,LDX,R
,IFT,SIZE,512,512
,STA,SIZE,1
,IFT,511,SIZE,SIZE
,STAE,SIZE,1
,LDX,L
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,LDX,R
,IFT,LINK,512,512
,STA,LINK,1
,IFT,511,LINK,LINK
,STAE,LINK,1
,JMP,$24
$23,EQU,*
,LDA,R
,STA,P
,LDA,L
,STA,R
$24,EQU,*
,LDX,L
```

```
[illegible]
,IFT,LINK,512,512
,LDA,LINK,1
,IFT,511,LINK,LINK
,LDAE,LINK,1
,STA,L
,JMP,$21
$22,EQU,*
,INCR,1
,STA,COMBINED
,JMP,RESA
,JMP*,$P4
$T4,EQU,*-1
,BSS,0
,JMP,$25
X,DATA,0
N,DATA,0
HEAD,DATA,0
,ORG,HEAD+2
C,DATA,0
B,DATA,0
FREE,DATA,0
COMBINED,DATA,0
NIL,DATA,077777,NIL
,ORG,NIL+2
S,DATA,0
R,DATA,0
R1,DATA,0
I,DATA,0
SINGLES,DATA,0
LF,DATA,0
SF,DATA,0
L,DATA,0
T,DATA,0
P,DATA,0
Q,DATA,0
U,DATA,0
$25,EQU,*
,END,$
```

## APPENDIX II--MOL620 COMPILER LISTING

```
.META PROGRAM (K=100,M=800,N=1500,S=400)
%           COMPILER FOR MOL620 PROGRAMS%
%           OUTPUTS DAS ASSEMBLY LANGUAGE %
%           STARTED NOVEMBER 8, 1970%
%           WRITTEN IN PDP-10 TREE-META VERSION 1.5%
%           PRODUCER: GREGORY L. HOPWOOD%
%           DIRECTOR: MARSHA DRAPKIN HOPWOOD%
%           CONSULTANT: WILLIAM M. NEWMAN%
%           ADAPTED FROM THE ORIGINAL MOL620 COMPILER%
%            WRITTEN IN FORTRAN FOR THE S/360 AND%
%            DESCRIBED IN UNIV OF CALIF, IRVINE%
%            INFO AND COMPUTER SCIENCE DEPARTMENT%
%            TECHNICAL MEMO #1, ENTITLED %
% "MOL620: A MACHINE ORIENTED LANGUAGE AND LANGUAGE COMPILER%
%        FOR THE VARIAN DATA 620/I COMPUTER" %
%            BY MARSHA A. DRAPKIN%
PROGRAM = HEAD IDORG $(UNIT) "DONE."
   ?"????  IN BEGIN...END BLOCK OR MISSING 'DONE.'"?
   [",END,$"\];
%PATCHES TREE-META TO USE '$' AS BASE FOR GENERATED LABELS%
HEAD = !( "HRR B,41"\"ADDI B,↑D20"\"MOVE A,[JFCL]"\"MOVEM A,(B)"\
          "MOVEM A,1(B)"\"MOVE A,[220000000000]"\"MOVEM A,4(B)"\)
          .EMPTY :VERSION[0]*;
IDORG = $(DEC* / COMMENT*) (←IDEN "ORG" VALUE '; :XORG[2] * /
                     .EMPTY ["$,EQU,*"\]) ;
UNIT = PROC / 1$(←DEC * / COMMENT *) ;
PROC = ← PRCHEAD FORMPART '; (COMSTMS / .EMPTY) "ENDP" ';
        [",JMP*,$P".WC\ "$T".WC+WC ",EQU,*-1"\ ",BSS,"?WA$WA\];
PRCHEAD = ←(←"PROCEDURE"/"PROC") IDEN
           ["$P".WC",EQU,*"\
  *SO",NOP,      <<<<<PROCEDURE " *SO ">>>>>"\];
FORMPART = ← '( 0$2(',) ') /
                 '( (IDEN [",STA,"*SO\] / .EMPTY)
                  (', (IDEN [",STB,"*SO\] / .EMPTY) / .EMPTY)
                  (', (IDEN [",STX,"*SO\] / .EMPTY) / .EMPTY) ') /
                  .EMPTY &;
COMSTMS = 1$(←DEC */ ←STMNT */ ←COMMENT*) ;
COMMENT = DAS '; ;
DEC = ←(DECLARE / ASSDEC) ;
ASSDEC = ←   (SETSTM / EQUSTM) '; ;
SETSTM = ← "SET" IITEM :XSET[2]  $(', IITEM :XSET[3] ) ;
EQUSTM = ← "EQU" IITEM :XEQU[2]  $(', IITEM :XEQU[3] );
IITEM = ← IDEN '=  VALUE ;
DECLARE = (←"DECLARE" /←"DECL") ITEM :XDEC[1] $(', ITEM :XDEC[2])
          '; :ZDEC[1];
ITEM = ← IDEN (←'[(←VALUE ':/.EMPTY :XMT[0]) VALUE '] /
               .EMPTY :XMT[0] :XMT[0] )
          ('= VALUE / .EMPTY :XZERO[0]) :XITEM[4]   ;
STMNT = ←(LABELID :XLABELID[1] / .EMPTY :XMT[0]) USTMNT '; :XSTMNT[2] ;
LABELID = ←.ID ': ;
```

```
USTMNT = IFS /
          FORS /
         WHILES /
         BLOCKS /
         ASSIGNS /
         NULLS /
         GOTOS /
         CALLS /
         RETURNS /
         BUMPS /
         STOPS /
         DAS ;
IFS = ←"IF" EXP :XEXP[1] "THEN" USTMNT
     ((←'; "ELSE" / "ELSE") USTMNT / .EMPTY :XMT[0]):XMT[0]:XIF[4];
FORS = ←"FOR" STOLOC :XDESIG[1] ('←/'=) EXP :XSTO[2]
      (("STEP" EXP:XSUM[0]:XSUB[0]:XSTEP[3]/.EMPTY:XMT[0])
       ("UNTIL" EXP :XLE[0]:XGE[0]:XUNTIL[3]/"WHILE" EXP))
       "DO" USTMNT :XFOR[4] ;
WHILES = ←"WHILE" EXP :XEXP[1] "DO" USTMNT :XMT[0]:XWHILE[3];
ASSIGNS =←DESIG ('← / '=) EXP :XSTO[2] ;
DESIG =←STOLOC :XDESIG[1] $(',STOLOC :XDESIG[2]) ;
NULLS = ←"NULL" :XMT[0];
BLOCKS = ←"BEGIN" BODY "END";
BODY = ← (DEC / STMNT / COMMENT) :XBOD[1]
       $((DEC / STMNT / COMMENT) :XBOD[2]) / .EMPTY :XMT[0];
GOTOS = ←"GO" "TO" JMPLOC :XGOTO[1];
CALLS = ←"CALL" JMPLOC (ARGLIST / .EMPTY :XMT[0]) :XCALL[2];
RETURNS = ←"RETURN" (ARGLIST / .EMPTY :XMT[0])
           ("FROM" JMPLOC / .EMPTY :XMT[0]) :XRET[2];
ARGLIST = ← '( (←EXP / .EMPTY :XMT[0]) :X1ARG[1]
          (←', (EXP  / .EMPTY :XMT[0]) / .EMPTY :XMT[0]) :X2ARG[1]
          (←', (EXP / .EMPTY :XMT[0]) / .EMPTY :XMT[0]) :X3ARG[1]
             ') :XARGLIST[3];
DAS = ← .SR  :XDAS[1]; %RECOGNIZE "STRING"%
BUMPS = ← "BUMP" BMPLOC :XBUMP[1] $(', BMPLOC :XBUMP[2] );
STOPS = ←("STOP" / "HALT") :XSTOP[0];
EXP =←INTRSECT $("XOR" INTRSECT :XXOR[2]/"OR" INTRSECT :XOR[2]);
INTRSECT = ← NEGATION $("AND" NEGATION :XAND[2]);
NEGATION = ←"NOT" NEGATION :XNOT[1] / ←RELATION ;
RELATION = ← SUM $(( ">=" :XGE[0] / '> :XGT[0] /
                       '= :XEQ[0] / "<=" :XLE[0] /
                       '< :XLT[0] / '# :XNE[0] ) SUM :ZREL[3]);
SUM = ← PRODUCT $(('+ :XSUM[0] / '- :XSUB[0]) PRODUCT :ZOPR[3]);
PRODUCT = ← FACTOR $(('* :XMUL[0]/ '/ :XDIV[0]) FACTOR :ZMUL[3]);
FACTOR = ←BITAND $((←"BOR" :XBOR[0] / "BXR" :XBXR[0]) BITAND :ZOPR[3]);
BITAND = ← SHIFT $("BAND" :XBAND[0] SHIFT :ZOPR[3] );
SHIFT = ← PRIMARY $('" IDEN '" VALUE :XSHIFT[3] );
PRIMARY = ←IDEN '[ EXP '] :XMT[0] :XARRAY[3] /
          ←JMPLOC ARGLIST :XCALL[2] /
          ←IDEN :XADDR[1] /
       ←'( (REGISTER / SWITCH) ') :XREG[1] /
       '( EXP ') :XEXP[1] /
       ← CONSTANT /
       ←'[ EXP '] :ZINDIR[1] /
       ← '- '- PRIMARY /
       '- PRIMARY :XPMINUS[1] /
       '+ PRIMARY /
       '@ IDEN :XREF[1] ;
```

```
CONSTANT = NUMBER /
            ← '- '- CONSTANT /
            %RECOGNIZE "STRING"%
            ←.SR :XADDR[1] /
             %THE NEXT ALTERNATIVE RECOGNIZES 'STRING'%
             ←'' .CHR .CHR '' :XALPHA[2] / %AS IN 'XY'%
             ←'' .CHR '' :XALPHA[1] / %AS IN 'X'%
             ←'[ (IDEN :XADDR[1] / NUMBER) '] :XINDIR[1] ;
REGISTER = ←"AR" :XAR[0] / ←"BR" :XBR[0] / ←"XR" :XXR[0] /
            ←"OF" :XOF[0] ;
SWITCH = ←"SS1" :XSS1[0] / ←"SS2" :XSS2[0] / ←"SS3" :XSS3[0] ;
STOLOC = ←IDEN ('[ EXP '] :XDESIG[0] :XARRAY[3] /
             .EMPTY :XADDR[1]) /
          ←'( REGISTER ') / ←CONSTANT / ←'[ EXP '] :ZINDIR[1] ;
BMPLOC = ←IDEN ('[ EXP '] :XBUMP[0] :XARRAY[3]/
             .EMPTY :XADDR[1]) /
          ← '( REGISTER ') / ←CONSTANT / ←'[ EXP '] :ZINDIR[1] ;
JMPLOC = ←IDEN ('[ EXP '] :XMT[0] :XARRAY[3] / .EMPTY :XADDR[1]) /
         ←CONSTANT / '[ EXP '] :ZINDIR[1];
VALUE =←IDEN :XNUM[1] / ←CONSTANT/ ←'- '- VALUE / ←'- IDEN :XMINUS[1];

IDEN = .ID;
NUMBER = .NUM / ←'- .NUM :XMINUS[1] / ←'+ .NUM;
%%
%%
%%
%%
%%
%%
%UNPARSE RULES FOLLOW%
%%
%%
%%
VERSION[] => $WA$WC <"MOL620  VERSION 2.15  (17-SEPT-1971)"\>\
          "* COMPILED BY MOL620  VERSION 2.15  (17-SEPT-71)"\ ",SMRY,"\;
XORG[-,-] =>  *1 ",ORG," *2\ "$,EQU,*"\ ;
XDAS[-] => *1 \;
XSTO[-,-] => YLDA[*2] *1;
ZDEC[-] => ",JMP,"#1\ *1 #1",EQU,*"\;
XDEC[-,-] => *1 XDEC[*2]
     [-] => *1;
XSET[-,-] => *1 ",SET," *2  \
     [-,-,-] => *1 XSET[*2,*3] ;
XEQU[-,-] => *1 ",EQU," *2  \
     [-,-,-] => *1 XEQU[*2,*3] ;
XDESIG[-,-] => XDESIG[*2] *1
   [XARRAY[-,-,-]] => (YTYPE[*1:*2] / ZARRAY[*1:*2] /
                        XMT[*1] ",STA,$T".WC"+"+WA.WA\) *1
       [XMT[]] => .EMPTY
       [XAR[]] => .EMPTY
       [XBR[]] => ",TAB,"\
       [XXR[]] => ",TAX,"\
       [XOF[]] => ",ROF,"\ ",JAZ,*+3"\ ",SOF,"\
       [ZINDIR[-]] => ",STA,$T".WC"+"+WA.WA\ YLDA[*1:*1]
                      ",STA,$T".WC"+"+WA.WA-WA\",LDA,$T".WC"+".WA\
                      ",STAE,($T".WC"+"+WA.WA-WA-WA")*"\
       [XINDIR[-]] => ",STAE," *1\
       [-] => ",STA," *1 \;
XITEM[-,XMT[],XMT[],-] => *1 ",DATA," *4 \
      [-,XMT[],-,-] => *1 ",DATA," *4\ ",ORG," *1 '+ *3\
      [-,.NUM,.NUM,-] => *1 ",EQU,*-" *2\",DATA," *4\
                         ",ORG," *1 '+ *3 "+1"\
       [-,XMINUS[-],.NUM,-] => *1 ",EQU,*+" *2:*1\
                 ",DATA," *4\ ",ORG,"*1 '+ *3 "+1"\
        [-,.NUM,XMINUS[-],-] => XITEM[*1,*3,*2,*4]
        [-,XMINUS[-],XMINUS[-],-] => *1 ",EQU,*+" *2:*1\
                       ",DATA," *4\ ",ORG," *1 '- *3:*1\
        [-,-,-,-] => <"ERROR IN DECLARATION OF " *1\>;
   XBUMP[XAR[]] => ",IAR,"\
        [XBR[]] => ",IBR,"\
        [XXR[]] => ",IXR,"\
        [XOF[]] => ",SOF,"\
        [XARRAY[-,-,-]] => *1
        [ZINDIR[-]] => YLDA[*1:*1] ",STA,$T".WC"+"+WA.WA\
                         ",INRE,($T".WC"+".WA-WA")*"\
        [-,-] => *1 XBUMP[*2]
        [XINDIR[-]] => ",INRE," *1\
        [-] => ",INR," *1 \;
   XZERO[] => '0 ;
   XMT /=> .EMPTY;
   XARGLIST[-,-,-] => YTYPE[*1:*1](YTYPE[*2:*1](YTYPE[*3:*1] *1 *2 *3
                        / *3:*1 ",TAX,"\ *1 *2)
                        /YTYPE[*3:*1] *2:*1 ",TAB,"\ *1 *3
                        /*2:*1 ",STA,$T".WC"+"+WA.WA\*3:*1 ",TAX,"\
                          ",LDB,$T".WC"+".WA-WA\ *1)
              /YTYPE[*2:*1] (YTYPE[*3:*1] *1:*1 *2 *3 / *1:*1
        ",STA,$T".WC"+"+WA.WA\*3:*1 ",TAX,"\",LDA,$T".WC"+".WA-WA\ *2)
             /YTYPE[*3:*1] *1:*1 ",STA,$T".WC"+"+WA.WA\
              *2:*1 ",TAB,"\ *3 ",LDA,$T".WC"+".WA-WA\
             /*1:*1 ",STA,$T".WC"+"+WA.WA\
              *2:*1 ",STA,$T".WC"+"+WA.WA\
              *3:*1 ",TAX,"\
              ",LDB,$T".WC"+".WA-WA\ ",LDA,$T".WC"+".WA-WA\;
   X1ARG[XMT[]] => .EMPTY
        [-] => YLDA[*1];
   X2ARG[XMT[]] => .EMPTY
        [-] => ",LDB" YMODE[*1]\ ;
   X3ARG[XMT[]] => .EMPTY
        [-] => ",LDX" YMODE[*1]\ ;
   XNUM[-] => *1;
   XMINUS[XMINUS[-]] => *1:*1
         [-] => '- *1;
   XPMINUS[XPMINUS[-]] => YLDA[*1:*1]
          [XMINUS[-]] => YLDA[*1:*1]
          [-] => YLDA[*1]  ",CPA,"\ ",IAR,"\;
   XALPHA[-] => "'" *1:C "'"
         [-,-] => "'" *1:C *2:C "'" ;
   XINDIR[-] => '( *1 ")*" ;
   ZINDIR[-] => YLDA[*1] ",STA,$T".WC"+"+WA.WA\",LDAE,($T".WC"+"
                 .WA-WA ")*" \;
   XADDR[-] => *1;
   XARRAY
      [-,ZOPR[-,XSUM[],"1"],-] => ZARRAY[*2] ",LDX" YMODE[*2:*1]\ ",IXR,"\
           ZARRAY[*1,*3] / ZARRAY[*1,*2,*3]
      [-,ZOPR[-,XSUB[],"1"],-] => ZARRAY[*2] ",LDX" YMODE[*2:*1] \
           ",DXR," \ ZARRAY[*1,*3] / ZARRAY[*1,*2,*3]
      [-,XARRAY[-,-,-],-] => ZARRAY[*2] ",LDX" YMODE[*2:*2]\
           ",IFT," *2:*1 ",512,512"\ ",LDX," *2:*1 ",1"\
           ",IFT,511," *2:*1 ', *2:*1\ ",LDXE," *2:*1 ",1"\
              ZARRAY[*1,*3] / ZARRAY[*1,*2,*3]
      [-,-,-] => ZARRAY[*1,*2,*3];
   YARRAY[-,-,XDESIG[]] => (YTYPE[*2] / XMT[*3] ",LDA,$T".WC"+".WA-WA\)
           ",IFT," *1 ",512,512"\ ",STA," *1 ",1"\
           ",IFT,511," *1 ', *1\ ",STAE," *1 ",1"\
        [-,-,XBUMP[]] => ",IFT," *1 ",512,512"\ ",INR," *1 ",1"\
         ",IFT,511," *1 ', *1\ ",INRE," *1 ",1"\
        [-,-,-] => ",IFT," *1 ",512,512"\ ",LDA," *1 ",1"\
          ",IFT,511," *1 ', *1\ ",LDAE," *1 ",1"\;
   ZARRAY[ZOPR[-,XSUM[],"1"]] => YTYPE[*1:*1]
          [ZOPR[-,XSUB[],"1"]] => YTYPE[*1:*1]
          [-,XDESIG[]]=> ",IFT," *1 ",512,512"\ ",STA," *1 ",1"\
            ",IFT,511," *1 ', *1\ ",STAE," *1 ",1"\
          [XARRAY[-,-,-]] => YTYPE[*1:*2]
          [-,XBUMP[]] => ",IFT," *1 ",512,512"\ ",INR," *1 ",1"\
            ",IFT,511," *1 ', *1\ ",INRE," *1 ",1"\
          [-,-] => ",IFT," *1 ",512,512"\ ",LDA," *1 ",1"\
            ",IFT,511," *1 ', *1\ ",LDAE," *1 ",1"\
          [-,-,-] => YTYPE[*2] ",LDX" YMODE[*2]\ YARRAY[*1,*2,*3] /
                 *2 ",TAX," \ YARRAY[*1,*2,*3] ;
   XREF[-] => *1;
   XREG[-] => *1 ;
   XAR[] => .EMPTY;
   XBR[] => ",TBA,"\;
   XXR[] => ",TXA,"\;
   XSHIFT[-,-,-] => YLDA[*1] ', *2 ', *3 \;
   XEXP[-] => YLDA[*1] ;
   XLABELID[-] => *1 ",EQU,*"\ ;
   XSTMNT[-,-] => *1 *2 ;
   XWHILE[XEXP[ZREL[-,-,-]],-,-] => #2 ",EQU,*"\
          ZREL[*1:*1:*1,*3,*1:*1:*3] XIF[*1:*1:*2] #1\
          *2 ",JMP," #2\ #1 ",EQU,*"\
          [-,-,-] => #2 ",EQU,*"\ *1 ",JAZ,"#1\ *2
                 ",JMP,"#2\ #1 ",EQU,*"\;
 ' XGOTO[XARRAY[-,-,-]] => YJMP[*1:*1,*1:*2] ",JMP*,$T".WC"+".WA-WA\
        [ZINDIR[-]] => ZJMP[*1:*1] ",JMP*,$T".WC"+".WA-WA\
        [-] => ",JMP," *1\;
   XCALL[XARRAY[-,-,-],-] => YJMP[*1:*1,*1:*2] *2 ",JMPM*,$T".WC"+".WA-WA\
        [ZINDIR[-],-] => ZJMP[*1:*1] *2 ",JMPM*,$T".WC"+".WA-WA\
        [-,-] => *2 ",JMPM," *1\;
   XRET[-,XMT[]] => *1 ",JMP*,$P" .WC\
       [-,XARRAY[-,-,-]] => YJMP[*2:*1,*2:*2] *1 ",JMP*,$T".WC"+".WA-WA\
       [-,ZINDIR[-]] => ZJMP[*2:*1] *1 ",JMP*,$T".WC"+".WA-WA\
       [-,-] => *1 ",JMP*," *2\;
   YJMP[-,-] => YLDA[*2] ",ADDI," *1\ ",STA,$T".WC"+"+WA.WA\;
   ZJMP[-] => YLDA[*1] ",STA,$T".WC"+"+WA.WA\;
   XSTOP[] => ",HLT,"\;
   XIF[XEXP[ZREL[-,-,-]],-,XMT[],-] => ZREL[*1:*1:*1,*4,*1:*1:*3]
          XIF[*1:*1:*2] #1 \ *2 #1 ",EQU,*"\
      [-,-,XMT[],-] => *1 ",JAZ,"#1\ *2 #1",EQU,*"\
      [XEXP[ZREL[-,-,-]],-,-,-] => ZREL[*1:*1:*1,*4,*1:*1:*3]
          XIF[*1:*1:*2] #1\ *2 ",JMP,"#2\ #1 ",EQU,*"\ *3 #2 ",EQU,*"\
      [-,-,-,-] => *1 ",JAZ,"#1\ *2 ",JMP,"#2\
                 #1",EQU,*"\ *3 #2",EQU,*"\
      [XGT[]] => ",DAR,"\ ",JAN,"
      [XGE[]] => ",JAN,"
      [XEQ[]] => ",JAZ,*+4"\ ",JMP,"
      [XNE[]] => ",JAZ,"
      [XLE[]] => ",DAR,"\ ",JAP,"
      [XLT[]] => ",JAP,";
   XFOR[-,-] => *1 *2
       [-,XMT[],XUNTIL[-,-,-],-] => *1 #1",EQU,*"\
         ZREL[*1:*1:*1,*3:*2,*3:*1] ",JAZ,"#2\
         *4 XBUMP[*1:*1:*1]
           ",JMP,"#1\ #2 ",EQU,*"\
       [-,XMT[],-,-] => *1 #1 ",EQU,*"\
         YLDA[*3] ",JAZ," #2\
         *4 XBUMP[*1:*1:*1]
         ",JMP," #1\ #2 ",EQU,*"\
        [-,-,XUNTIL[-,-,-],-] => *1 #1",EQU,*"\
          (YSTEP[*2:*1] ZREL[*1:*1:*1,*3:*2,*3:*1] ",JAZ,"#2\/
           ZSTEP[*2:*1] ZREL[*1:*1:*1,*3:*3,*3:*1] ",JAZ," #2\/
            ZOPR[*1:*1:*1,*2:*3,*3:*1]  %AR←(V-C)   A STEP B UNTIL C%
            ",STA,$T".WC"+"+WA.WA\
            YLDA[*2:*1] ",ASRA,15"\ ",ORAI,1"\ %SIGN OF B%
            ",TAB,"\ ",TZA,"\ ",MUL,$T".WC"+".WA-WA\ %(V-C)*SIGN(B)%
            ",JAZ,*+4"\ ",JAP," #2\ ) %CONTINUE IF AR<=0 %
         *4 ZOPR[*1:*1:*1,*2:*2,*2:*1] *1:*1
         ",JMP,"#1\ #2 ",EQU,*"\
        [-,-,-,-] => *1 #1",EQU,*"\
         YLDA[*3] ",JAZ," #2\
         *4 ZOPR[*1:*1:*1,*2:*2,*2:*1] *1:*1
         ",JMP,"#1\ #2 ",EQU,*"\ ;
    YSTEP[.NUM] => .EMPTY
         [XREF[-]] => .EMPTY;
    ZSTEP[XMINUS[-]] => .EMPTY;
    XSTEP /=> .EMPTY;
    XUNTIL /=> .EMPTY;
    XBOD[-,-] => *1 *2
        [-] => *1;
    YMODE[.NUM] => "I," *1
          [XMINUS[-]] => "I," *1
         [XADDR[-]] => ', *1
         [XREF[-]] => "I," *1
         [XALPHA[-]] => "I," *1
          [XALPHA[-,-]] => "I," *1
         [XINDIR[-]] => "E," *1;
    YTYPE[.NUM] => .EMPTY
         [XMINUS[-]] => .EMPTY
         [XADDR[-]] => .EMPTY
         [XREF[-]] => .EMPTY
         [XALPHA[-]] => .EMPTY
         [XALPHA[-,-]] => .EMPTY
         [XINDIR[-]] => .EMPTY
         [XNUM[-]] => .EMPTY
         [XMT[]] => .EMPTY ;
    YABELIAN[XSUM[]]=> .EMPTY
             [XBAND[]] => .EMPTY
             [XBOR[]] => .EMPTY
             [XBXR[]] => .EMPTY;
    YLDA["0"] => ",TZA,"\
        ["1"] => ",INCR,1"\
        [XMINUS["1"]] => ",DECR,1"\
        [-] => YTYPE[*1] ",LDA" YMODE[*1]\ / *1 ;
    XBAND[] => ",ANA" ;
    XBOR[] => ",ORA";
    XBXR [] => ",ERA";
    XSUM[] => ",ADD";
    XSUB[] => ",SUB";
    XGT[] => ",DAR,"\",CPA,"\",LSRA,15"\;
    XGE[] => ",CPA,"\",LSRA,15"\;
    XEQ[] => ",JAZ,*+3"\",DECR,1"\",IAR,"\;
    XLE[] => ",DAR,"\",LSRA,15"\;
    XLT[] => ",LSRA,15"\;
    XNE[] => .EMPTY;
    XMUL[] => .EMPTY;
    XDIV[] => .EMPTY;
    XNOT[XNOT[-]] => *1:*1
        [-] => YLDA[*1] XEQ[] ;
```

```
XOF[] => ",TZA,"\",AOFA,"\;
XSS1[] => ",INCR,1"\",JSS1,*+3"\",TZA,"\;
XSS2[] => ",INCR,1"\",JSS2,*+3"\",TZA,"\;
XSS3[] => ",INCR,1"\",JSS3,*+3"\",TZA,"\;
XAND[-,-] => YLDA[*1] ",JAZ," #1\ YLDA[*2] #1 ",EQU,*"\;
XXOR[-,-] => YLDA[*2] ",STA,$T".WC"+"+WA.WA\ YLDA[*1]
             ",JAZ,*+8"\",LDB,$T".WC"+".WA\",JBZ,*+5"\
             ",TZA,"\",JMP,*+3"\",ERA,$T".WC"+".WA-WA\;
XOR[-,-] => YLDA[*1] ",JAZ,*+4"\ ",JMP,"#1\ YLDA[*2] #1 ",EQU,*"\;
ZREL[-,-,"0"] => YLDA[*1] *2
    [-,-,-] => (YTYPE[*1](YTYPE[*3] ",LDA" YMODE[*1]\",SUB" YMODE[*3]\/
                            *3 ",STA,$T".WC"+"+WA.WA\",LDA"YMODE[*1]\
                            ",SUB,$T".WC"+".WA-WA\ ) /
                  YTYPE[*3] *1 ",SUB" YMODE[*3]\ /
         *3 ",STA,$T".WC"+"+WA.WA\*1 ",SUB,$T".WC"+".WA-WA\ ) *2 ;
ZOPR[-,XSUM[],"1"] => YLDA[*1] ",IAR,"\
    [-,XSUB[],"1"] => YLDA[*1] ",DAR,"\
    [-,-,-] => YTYPE[*1](YTYPE[*3] YLDA[*1] *2 YMODE[*3]\/
                    YABELIAN[*2] *3 *2 YMODE[*1] \/
                    *3 ",STA,$T".WC"+"+WA.WA \ YLDA[*1]
                    *2 ",$T".WC"+".WA-WA\ ) /
                YTYPE[*3] *1 *2 YMODE[*3] \ /
          YABELIAN[*2] *1 ",STA,$T".WC '+ +WA.WA\
                *3 *2 ",$T".WC '+ .WA-WA\ /
                *3 ",STA,$T".WC"+"+WA.WA\ *1 *2 ",$T".WC"+".WA-WA\ ;
ZMUL[-,XMUL[],"2"] => YLDA[*1] ",ASLA,1"\
    [-,XMUL[],"4"] => YLDA[*1] ",ASLA,2"\
    [-,XMUL[],"8"] => YLDA[*1] ",ASLA,3"\
    [-,XMUL[],"16"]=> YLDA[*1] ",ASLA,4"\
    [-,XMUL[],"256"]=> YLDA[*1] ",ASLA,8"\
    [-,XMUL[],-] => YLDA[*1] ",TAB,"\",TZA,"\
                       (YTYPE[*3] ",MUL" YMODE[*3] \/
    ",STA,$T".WC"+"+WA.WA\ *3 ",TAB,"\ ",TZA,"\ ",MUL,$T".WC"+".WA-WA\)
                ",LASL,15"\
    [-,XDIV[],"2"] => YLDA[*1] ",TAB,"\ ",LASR,1"\ ",ASRB,14"\
    [-,XDIV[],-] => YTYPE[*3] (YLDA[*1]
                        ",LASR,15"\ ",DIV" YMODE[*3]\ YDV2[]) /
                       *3 ",STA,$T".WC"+"+WA.WA\
                      YLDA[*1] ",LASR,15"\
                      ",DIV,$T".WC"+".WA-WA\ YDV2[] ;
YDV2[] => ",TAX,"\ ",TBA,"\ ",TXB,"\ ;
.END
```