BLISS/11 REFERENCE MANUAL

A BASIC LANGUAGE FOR THE IMPLEMENTATION OF
SYSTEM SOFTWARE FOR THE PDP-11

W. WULF
J. APPERSON
R. BRENDER ☆
C. GESCHKE
P. KNUEVEN ☆
C. WEINSTOCK
J. ZARRELLA
D. WILE

COMPUTER SCIENCE DEPARTMENT
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA

MARCH 1, 1972

☆ DIGITAL - EQUIPMENT CORP, MAYNARD MASS. (01754)

THIS WORK WAS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY OF THE OFFICE OF THE SECRETARY OF DEFENSE (F44620-70-C-0107) AND IS MONITORED BY THE AIR FORCE OFFICE OF SCIENTIFIC RESEARCH. THIS DOCUMENT HAS BEEN APPROVED FOR PUBLIC RELEASE AND SALE; ITS DISTRIBUTION IS UNLIMITED.

CONTENTS

I. LANGUAGE DEFINITION

I.1.1 MODULES
I.1.2 BLOCKS AND COMMENTS
I.1.3 LITERALS
I.1.4 NAMES
I.1.5 THE "CONTENTS OF" OPERATOR
I.1.6 POSITION AND SIZE MODIFIER
I.1.7 DEFAULT POSITION AND SIZE
I.1.8 DATA REPRESENTATION
I.1.9 EXAMPLES
I.2.1 EXPRESSIONS
I.2.2 SIMPLE EXPRESSIONS
I.2.3.1 CONTROL EXPRESSIONS
I.2.3.2 CONDITIONAL EXPRESSIONS
I.2.3.3 LOOP EXPRESSIONS
I.2.3.4 ESCAPE EXPRESSIONS
I.2.3.5 CHOICE EXPRESSIONS
I.2.3.6 CO-ROUTINE EXPRESSIONS
I.3.1 DECLARATIONS
I.3.1.1 STORAGE (AN INTRODUCTION)
I.3.1.2 STRUCTURE (AN INTRODUCTION)
I.3.1.3 ACTUAL DECLARATION SYNTAX
I.3.2 MEMORY ALLOCATION
I.3.3 MAP DECLARATION
I.3.4 BIND DECLARATIONS
I.3.5 STRUCTURES
I.3.6 ROUTINES
I.3.7 LABEL DECARATION
I.3.8 MACROS
I.3.9 UNDECLARE DECLARATION
I.3.10 CALLING SEQUENCE DECLARATIONS
I.3.11 LINKAGE DECLARATION
I.3.12 SIGNAL EXPRESSION

II. SPECIAL LANGUAGE FEATURES

II.1.1 SPECIAL FUNCTIONS
II.1.2 MACHINE LANGUAGE

III.SYSTEM FEATURES

III.1.0 COMPILATION CONTROL
III.1.1 COMMAND SYNTAX
III.1.2 MODULE HEAD
III.1.3 SWITCHES DECLARATION
III.1.4 ACTIONS

III.2 ERROR REPORTING

INTRODUCTION

BLISS-11 IS A PROGRAMMING LANGUAGE FOR THE PDP-11. IT IS SPECIFICALLY INTENDED TO BE USED FOR IMPLEMENTING "SYSTEM SOFTWARE". AS SUCH IT DIFFERS FROM OTHER LANGUAGES IN SEVERAL SIGNIFICANT WAYS.

1. HIGHER LEVEL LANGUAGES DERIVE THEIR SUITABILITY FOR A PARTICUALR PROBLEM AREA - FORTRAN/ALGOL FOR MATHEMATICS, SNOBOL FOR STRINGS, GPSS FOR SIMULATIONS, ETC - BECAUSE THEY PROVIDE TO THE USER A VOCABULARLY AND SET OF CONVENTIONS APPROPRIATE TO THAT PROBLEM AREA. WE VIEW "IMPLEMENTATION LANGUAGES" IN A SIMILAR WAY - AS APPLICATION LANGUAGES WHERE THE APPLICATION IS A PARTICULAR BRAND OF HARDWARE. AS SUCH, AN IMPLEMENTATION LANGUAGE MUST REFLECT THE CAPABILITIES AND ARCHITECTURE OF ITS MACHINE.

2. INPUT/OUTPUT IS NOT A PART OF BLISS-11. I/O CAN BE DONE DIRECTLY IN THE LANGUAGE MUCH AS AN ASSEMBLY PROGRAM MIGHT, OR THROUGH SUBROUTINE CALLS.

3. EVERY ATTEMPT HAS BEEN MADE TO GIVE THE USER EXPLICIT CONTROL OVER THE CODE HE GENERATES AS HE CHOOSES, WHILE PROVIDING MAXIMUM CONVENIENCE OTHERWISE.

4. THERE ARE NO EXPLICIT OR IMPLICIT DATA MODES OTHER THAN THE 16 BIT BINARY WORD. DATA MODES ARE ESSENTIALLY USER DEFINED VIA THE STRUCTURE MECHANISM WHICH ALLOWS THE USER TO SET OR COMPUTE AN ALGORITHM FOR DATA ACCESS.

5. CLEARLY BLISS-11 IS A CLOSE RELATIVE OF BLISS-10 FOR THE PDP-10 AND SHARES MANY OF THAT LANGUAGES BIASES TOWARD TREATMENT OF IDENTIFIERS, LACK OF GOTOES, IMPORTANCE OF STRUCTURES, ETC. THE INTERESTED READER MAY FIND IT USEFUL TO EXAMINE THE DIFFERENCES BETWEEN THE TWO LANGUAGES AND HOW THESE ARE A REFLECTION OF THE DIFFERENCES IN THE MACHINES THEMSELVES.

I. LANGUAGE DEFINITION

I.1.1 MODULES

A MODULE IS A PROGRAM ELEMENT WHICH MAY BE COMPILED INDEPENDENTLY OF OTHER ELEMENTS AND SUBSEQUENTLY LOADED WITH THEM TO FORM A COMPLETE PROGRAM.

```
MODULE    ::=   BLOCK/
                MODULE-HEAD BLOCK 'ELUDOM'
MODULE-HEAD::=  'MODULE' NAME (PARAMETERS)=
```

A MODULE MAY REQUEST ACCESS TO OTHER MODULES' VARIABLES AND FUNCTIONS BY DECLARING THEIR NAMES IN EXTERNAL DECLARATIONS. A MODULE PERMITS GENERAL USE OF ITS OWN VARIABLES AND ROUTINES BY MEANS OF GLOBAL DECLARATIONS. THESE LINES OF COMMUNICATION BETWEEN MODULES ARE LINKED BY THE LOADER PRIOR TO EXECUTION. A COMPLETE PROGRAM CONSISTS OF A SET OF COMPILED MODULES LINKED BY THE LOADER. WHEN LOADING A NUMBER OF MODULES, IT IS NECESSARY THAT AT LEAST ONE MODULE CONTAIN A MODULE HEAD. THIS IS REQUIRED TO SET UP A STACK. NOTE THAT THE ELUDOM TO TERMINATE THE MODULE IS OPTIONAL. ALL THAT IS REQUIRED IS THAT THE BLOCK WHICH BEGINS AFTER THE MODULE DECLARATION IS CORRECTLY TERMINATED.

THE <PARAMETERS> FIELD OF A MODULE DEFINITION IS USED TO CONTROL THE COMPILATION. SEE SECTION III.1.3 FOR A DESCRIPTIVE LIST AND DEFAULTS.

EXAMPLE:

```
!THE START MODULE FOR BLISS IS COMPILED IN FRONT OF A
!NUMBER OF OTHER BLISS MODULES.

MODULE START (STACK=#40,SYNTAX)=
BEGIN
    .
    .
    .
END
ELUDOM          !THIS LINE IS OPTIONAL
```

IT IS CONVENIENT TO LEAVE OUT THE ELUDOM IF, FOR EXAMPLE, YOU WISH TO ADD A PARTICULAR MODULE HEAD TO AN ALREADY EXISTING BLOCK. THE SOURCE FILE FOR THE MODULE HEAD SHOULD PRECEDE THE BLOCK FILE IN YOUR COMMAND STRING AND THE TERMINAL ELUDOM NEED NOT BE APPENDED AT THE END.

I.1.2 BLOCKS AND COMMENTS

A BLOCK IS AN ARBITRARY NUMBER OF DECLARATIONS FOLLOWED BY AN ARBITRARY NUMBER OF EXPRESSIONS ALL SEPARATED BY SEMICOLONS AND ENCLOSED IN A MATCHING BEGIN-END OR "(" - ")" PAIR.

```
BLOCK ::= 'BEGIN' BLOCKBODY 'END' / ( BLOCKBODY )
BLOCKBODY ::= DECLARATIONS EXPRESSIONS
DECLARATIONS ::= /DECLARATION;/
                    DECLARATIONS; DECLARATION;
COMPOUNDEXPRESSION ::= 'BEGIN' EXPRESSIONS 'END' /
                         ( EXPRESSIONS )
EXPRESSIONS ::= /E/LABEL:E/ E; EXPRESSIONS
COMMENT ::= / ! RESTOFLINE ENDOFLINESYMBOL/
              % STRINGWITHNOPERCENT %
```

COMMENTS MAY BE ENCLOSED BETWEEN THE SYMBOL ! AND THE END OF THE LINE ON WHICH THE ! APPEARS. HOWEVER, A ! MAY APPEAR IN THE QUOTED STRING OF A LITERAL, OR BETWEEN TWO % SYMBOLS, WITHOUT BEING CONSIDERED THE BEGINNING OF A COMMENT. LIKEWISE, A % ENCLOSED WITHIN QUOTES WILL BE CONSIDERED PART OF A STRING.

AS IN ALGOL THE BLOCK INDICATES THE LEXICAL SCOPE OF THE NAMES DECLARED AT ITS HEAD. HOWEVER, IN CONTRAST TO ALGOL, THERE IS AN EXCEPTION. THE NAMES OF GLOBAL VARIABLES AND ROUTINES HAVE A SCOPE BEYOND THE BLOCK AND ALTHOUGH THEY ARE DECLARED WITHIN THE MODULE, THE EFFECT, FOR A MODULE CITING THEM IN AN EXTERMAL DECLARATION, IS AS IF THEY WERE DECLARED IN THE CURRENT BLOCK.

I.1.3 LITERALS

TWO BASIC DATA ELEMENTS ARE RECOGNIZED FOR THE PDP-11: THE 16 BIT WORD AND THE 8 BIT BYTE. LITERALS ARE NORMALLY CONVERTED TO A SINGLE WORD, BUT MAY IN FACT END UP IN EITHER OF THE TWO FORMS DEPENDING ON USAGE.

```
LITERAL ::= NUMBER/QUOTEDSTRING/PLIT
NUMBER ::= DECIMAL/OCTAL
DECIMAL ::= DIGIT/DECIMAL DIGIT
OCTAL ::= #OIT/OCTAL OIT
OIT ::= 0/1/2 --- /7
DIGIT ::= 0/1/2 --- /9
```

NUMBERS (UNSIGNED INTEGERS) ARE CONVERTED TO BINARY MODULO THE DATA SIZE. THE BINARY NUMBER IS 2'S COMPLEMENT AND IS SIGNED. OCTAL CONSTANTS ARE PREFIXED BY THE SHARP SIGN, #.

```
QUOTEDSTRING ::= "STRING"/'STRING'
```

QUOTED-STRING LITERALS MAY BE USED TO SPECIFY BIT PATTERNS CORRESPONDING TO THE 8-BIT ASCII CODE FOR PRINTING GRAPHIC CHARACTERS ON THE EXTERNAL I/O MEDIA. STRINGS OF ONE OR TWO CHARACTERS MAY BE USED FREELY AS CHARACTER CONSTANTS. LONGER STRINGS MAY BE USED IN OWN AND GLOBAL DECLARATIONS AS VALUES FOR THE NAMESIZEVALUE (SEE SECTION 1.3.2). STRINGS ENCLOSED IN DOUBLE QUOTE WILL ALWAYS BE RIGHT JUSTIFIED WITH THE FIRST BYTE OF THE FIRST WORD 0 IN THE CASE OF STRINGS WITH AN ODD NUMBER OF CHARACTERS.

WITHIN A QUOTED STRING THE QUOTING CHARACTER " IS REPRESENTED BY TWO SUCCESSIVE OCCURRENCES OF THAT CHARACTER.

I.1.3.1 POINTERS TO LITERALS

A PLIT IS A POINTER TO A LITERAL WORD WHOSE CONTENTS ARE SPECIFIED AT COMPILE TIME; E.G., PLIT 3 IS A POINTER TO A WORD WHOSE CONTENTS WILL BE SET TO 3 AT LOAD TIME.

```
PLIT ::= PLIT PLITARG
PLITARG ::= LOAD-TIME-EXPRESSION /
            LONG-STRING /
            TRIPLE
TRIPLE ::= (TRIPLE-ITEM-LIST)
TRIPLE-ITEM-LIST ::= TRIPLE-ITEM /  TRIPLE-ITEM, TRIPLE-ITEM-LIST
TRIPLE-ITEM ::= LOAD-TIME-EXPRESSION /
                LONG-STRING /
                DUPLICATION-FACTOR: PLITARG
DUPLICATION-FACTOR ::= COMPILE-TIME-EXPRESSION
```

*NOTE: "PLIT (3)+4" HAS 2 PARSES: PLIT LOAD-TIME-EXPRESSION
AND PLIT TRIPLE + EXPRESSION

THE LATTER CHOICE IS USED. HENCE, "PLIT (3)+4" IS THE SAME AS "(PLIT 3)+4".

A PLIT MAY POINT TO A CONTIGUOUSLY STORED SEQUENCE OF LITERALS - LONG STRINGS AND NESTED LISTS OF LITERALS ARE ALSO ALLOWED. THE VALUE OF

```
PLIT (3,5,7,9)
```

IS A POINTER TO 4 CONTIGUOUS WORDS CONTAINING 3,5,7 AND 9 RESPECTIVELY. A LONG STRING (> 2 CHARACTERS) IS ALSO A VALID ARGUMENT TO A PLIT:

```
PLIT "THIS ALLOCATES 12 WORDS"
```

ALLOCATES 12 WORDS OF 8-BIT ASCII CHARACTERS.

THE ARGUMENTS TO PLITS NEED ONLY BE CONSTANT AT LOAD TIME; PLITS ARE THEMSELVES LITERALS, THUS NESTING OF PLITS IS ALLOWED (WITH THE INNER PLITS ALLOCATED FIRST):

```
EXTERNAL A,B,C;
BIND Y = PLIT (A, PLIT (B,C), PLIT 3, "A LONG STRING", 5+9*3);
```

IS SUCH THAT:

```
.Y[0] = A; ..Y[1] = B; .(.Y[1]+1) = C
..Y[2] = 3; .Y[3] = "A "; .Y[4] = "LO"; .Y[5] = "NG";
.Y[10] = 32;
```

IN ADDITION, ANY ARGUMENT TO A PLIT CAN BE REPLICATED BY SPECIFYING THE NUMBER OF TIMES IT IS TO BE REPEATED; E.G.

```
PLIT (7:3)
```

PRODUCES A POINTER TO 7 CONTIGUOUS WORDS, EACH OF WHICH CONTAINS THE VALUE 3. DUPLICATED PLITS ARE ALLOCATED ONCE, IDENTICAL PLITS ARE NOT POOLED - HENCE,

```
BIND X = PLIT (3; PLIT A, PLIT A, 2; (2,3));
```

IS SUCH THAT:

```
..X[0] = ..X[1] = ..X[2] = ..X[3] = A;
 .X[0] =  .X[1] =  .X[2] NOT = .X[3];
 .X[4] =  .X[6] = 2; .X[5] = .X[7] = 3;
```

NOTE: THE LENGTH OF EVERY PLIT (IN WORDS) IS STORED AS THE WORD PRECEDING THE PLIT. HENCE, IN THE LAST EXAMPLE, .X[-1] = 8.

I.1.4 NAMES

SYNTACTICALLY AN IDENTIFIER, OR NAME, IS COMPOSED OF A SEQUENCE OF LETTERS AND/OR DIGITS, THE FIRST OF WHICH MUST BE A LETTER. CERTAIN NAMES ARE RESERVED AS DELIMITERS, SEE APPENDIX A. SEMANTICALLY THE OCCURRENCE OF A NAME IS EXACTLY EQUIVALENT TO THE OCCURRENCE OF A POINTER TO THE NAMED ITEM. THE TERM "POINTER" WILL TAKE ON SPECIAL CONNOTATION LATER WITH RESPECT TO CONTIGUOUS SUB-FIELDS WITHIN A WORD; HOWEVER, FOR THE PRESENT DISCUSSION THE TERM MAY BE EQUATED WITH "ADDRESS". THIS INTERPRETATION OF NAME IS UNIFORM THROUGHOUT THE LANGUAGE AND THERE IS NO DISTINCTION BETWEEN LEFT AND RIGHT HAND VALUES. CONTRAST THIS WITH ALGOL WHERE A NAME USUALLY, BUT NOT ALWAYS, MEANS "CONTENTS OF".

THE POINTER INTERPRETATION REQUIRES A "CONTENTS OF" OPERATOR, AND "." HAS BEEN CHOSEN. THUS .A MEANS "CONTENTS OF LOCATION A" AND ..A MEANS "CONTENTS OF THE LOCATION WHOSE NAME IS STORED IN LOCATION A". TO ILLUSTRATE THE CONCEPT, CONSIDER THE ASSIGNMENT EXPRESSION

```
P = E
```

THIS MEANS "STORE THE VALUE COMPUTED FROM E INTO THE LOCATION WHOSE POINTER IS THE VALUE OF P". (FURTHER DETAILS ARE GIVEN IN I.2.2.) THUS THE ALGOL STATEMENT "A := B" IS WRITTEN "A=.B". IT IS IMPOSSIBLE TO EXPRESS IN ALGOL BLISS EXPRESSIONS SUCH AS: "A=B", "A=..B", ".A=.B", ETC.

NOTE: BLISS-11 CURRENTLY ACCEPTS THE LEFT-ARROW CHARACTER AS THE ASSIGNMENT OPERATOR AS WELL AS THE EQUAL SIGN AS DESCRIBED IN THIS MANUAL.

THE FOLLOWING IDENTIFIERS ARE INITIALLY DECLARED IN THE BLISS-11 COMPILER:

```
R0 - REGISTER 0
R1 - REGISTER 1
R2 - REGISTER 2
R3 - REGISTER 3
R4 - REGISTER 4
R5 - REGISTER 5
DP - REGISTER 5: THE DISPLAY REGISTER
SP - REGISTER 6: THE STACK REGISTER
PC - REGISTER 7: THE PROGRAM COUNTER
VREG - REGISTER 0: THE VALUE REGISTER
```

I.1.5 THE "CONTENTS OF" OPERATOR

THE OPERATOR "." IS A UNARY OPERATOR USED TO DESIGNATE THE CONTENTS OF THE LOCATION NAMED BY ITS OPERAND. THAT LOCATION MAY BE IN CORE MEMORY OR ONE OF THE REGISTERS. THUS IF "X" IS THE NAME OF A WORD OF MEMORY, THEN ".X" NAMES ITS CONTENTS, AND "..X" NAMES THE CONTENTS OF THE WORD POINTED AT BY THE CONTENTS OF LOCATION X. SIMILARLY, IF R IS DECLARED A REGISTER NAME, THEN ".R" DESIGNATES THE CONTENTS OF THAT REGISTER.

I.1.6 POSITION AND SIZE MODIFIER

THE BLISS-11 LANGUAGE PROVIDES THE FOLLOWING NOTATION AS A MODIFIER TO EITHER THE "CONTENTS OF" OR ASSIGNMENT OPERATORS:

```
<POSITION,SIZE>
```

THE POSITION AND SIZE TOGETHER NAME A FIELD WHICH BEGINS "POSITION" BITS FROM THE RIGHT HAND SIDE OF THE NAMED BYTE AND EXTENDING "SIZE" BITS TO THE LEFT. THUS .X<0,8> DESIGNATES THE CONTENTS OF THE BYTE AT LOCATION X AND .X<0,16> DESIGNATES THE CONTENTS OF THE WORD AT LOCATION X.

ON THE PDP-11/20 THE SIZE MAY NOT EXCEED 16 BITS AND THE FIELD MAY NOT CROSS WORD BOUNDRIES.

WHEN USED WITH THE ASSIGNMENT OPERATOR THE NAMED FIELD IS USED AS THE DESTINATION OF THE RIGHT HAND SIDE. THE CONTENTS OF OTHER BITS IN A WORD ARE NOT AFFECTED.

EXAMPLES:

```
X=.Y<0,8>          ;
X=.Y<8,8>          ;
X=.(Y+1)<0,8>      ;!EQUIVALENT TO PREVIOUS LINE
X=.(Y-1)<16,8>     ;!EQUIVALENT TO PREVIOUS LINE
X<8,2>=1           ;
```

NOTE THAT THE POSITION AND SIZE NOTATION FUNCTIONS VERY SIMILIARLY TO THAT OF BLISS-10. HOWEVER, THE LEGAL BLISS-10 ASSIGNMENT

```
X=Y<3,2>           ;
```

HAS NO INTERPRETATION IN BLISS-11 SINCE THERE IS NO WAY TO TREAT THE POSITION AND SIZE AS DATA AT EXECUTION TIME. (A SYNTAX ERROR WILL BE REPORTED IN SUCH A CASE.) IN MOST CASES, THE RESULTING INTERPRETATION IS THE SAME IN BOTH LANGUAGES, HOWEVER.

SIMILARLY, NOTE THAT THE EXPRESSION

```
Y←.(IF .X THEN W<3,2> ELSE V<4,2>)
```

IS AN INCORRECT USAGE OF POINTERS IN BLISS-11.

NOTE THE SEMANTICS OF

```
.X<0,8>←.Y
```

DIFFERS FROM

```
(.X)<0,8>←.Y
```

IN THE FIRST CASE THE POSITION-SIZE MODIFIER IS ASSOCIATED WITH THE CONTENTS OPERATOR (".") WHILE IN THE SECOND CASE IT IS ASSOCIATED WITH THE ASSIGNMENT OPERATOR. SEE SECTION I.1.9 FOR FURTHER EXAMPLES.

I.1.7 DEFAULT POSITION AND SIZE

IN THE ABSENCE OF A POSITION-SIZE MODIFIER THE COMPILER ASSUMES A POSITION OF ZERO AND A SIZE EQUAL TO THAT GIVEN AT THE TIME OF ALLOCATION OF THE NAME. REGISTERS HAVE A SIZE OF 16 BITS. MEMORY HAS A SIZE OF EITHER 8 OR 16.

I.1.8 DATA REPRESENTATION

ALL WORD VALUES ARE NORMALLY TREATED AS TWOS-COMPLEMENT SIGNED INTEGERS. CERTAIN OPERATORS, EG. . (DOT), LEQU, ETC., TREAT THE WORD AS AN UNSIGNED INTEGER VALUE.

ALL BYTE QUANTITIES ARE NORMALLY TREATED AS UNSIGNED INTEGERS RANGING BETWEEN 0 AND 255 IN VALUE.

ALL BOOLEAN TESTS DEPEND ON THE LOW-ORDER BIT OF THE WORD OR FIELD BEING TESTED. THE RELATIONAL OPERATORS GENERATE A FULL WORD 0 OR 1 AS THEIR VALUE, WHEN REQUIRED TO YIELD A VALUE. ZERO REPRESENTS FALSITY, AND ONE REPRESENTS TRUTH.

I.1.9 EXAMPLES

THE FOLLOWING EXAMPLES OF BLISS-11 SOURCE TEXT AND THE CORRESPONDING MACHINE LANGUAGE EQUIVALENT IS OFFERRED TO ILLUSTRATE SOME OF THE IMPLICATIONS OF THE PRECEDING SECTIONS.

BLISS-11	PAL11
	(R,R1,R2 ARE REGISTER NAMES)
A = B;	MOV #B,A
A = .B;	MOV B,A
A <0,8>= B;	MOVB #B,A
A <0,8>= .B;	MOVB B,A
A <0,8>= .B<0,8>;	MOVB B,A
A = .B<0,8>;	CLR R BISB B,R MOV R,A
(.A)<0,8> = .B<8,8>;	MOV A,R MOVB B+1,@R
.A = .B;	MOV B,@A
.A<0,8> = .B;	CLR R BISB A,R MOV B,@R
(.A)<0,8> = .B	MOV A,R MOVB B,@R
A = ..B;	MOV @B,A
A = ..B<0,8>;	CLR R BISB B,R MOV @R,A

```
A = .(.B)<0,8>;          MOV     B,R
                         CLR     R1
                         BISB    @R,R1
                         MOV     R1,A

A = .(B+1)<0,8>;         CLR     R
                         BISB    B+1,R
                         MOV     R,A

A<0,8> = .(B+1)<0,8>;    MOVB    B+1,A

(A+1)<0,8> = .B<8,8>;    MOVB    B+1,A+1

(A+.C)<0,8> = .B<0,8>+1; CLR     R1
                         BISB    B,R1
                         INC     R1
                         MOV     C,R2
                         MOVB    R1,A(R2)
```

I.2.1 EXPRESSIONS

EVERY EXECUTABLE FORM IN THE BLISS LANGUAGE (THAT IS, EVERY FORM EXCEPT THE DECLARATIONS) COMPUTES A VALUE. THUS ALL COMMANDS ARE EXPRESSIONS AND THERE ARE NO "STATEMENTS" IN THE SENSE OF ALGOL OR FORTRAN. IN THE SYNTAX DESCRIPTION E IS USED AS AN ABBREVIATION FOR EXPRESSION.

E ::= SIMPLEEXPRESSION / CONTROLEXPRESSION

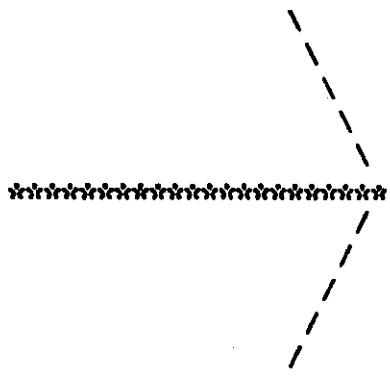
I.2.2 SIMPLE EXPRESSIONS

THE SEMANTICS OF SIMPLEEXPRESSIONS IS MOST EASILY DESCRIBED IN TERMS OF THE RELATIVE PRECEDENCE OF A SET OF OPERATORS, BUT READERS SHOULD ALSO REFER TO THE BNF-LIKE DESCRIPTION IN 4.1. THE PRECEDENCE NUMBER USED BELOW SHOULD BE VIEWED AS AN ORDINAL, SO THAT 1 MEANS FIRST AND 2 SECOND IN PRECEDENCE. IN THE FOLLOWING TABLE THE LETTER E HAS BEEN USED TO DENOTE AN ACTUAL EXPRESSION OF THE APPROPRIATE SYNTACTIC TYPE, SEE APPENDIX A.

PRECEDENCE	EXAMPLE	SEMANTICS
1	COMPOUNDEXPRESSION	THE COMPONENT EXPRESSIONS ARE EVALUATED FROM LEFT
1	BLOCK	TO RIGHT AND THE FINAL VALUE IS THAT OF THE LAST COMPONENT EXPRESSION.
1	E0(E1,E2,...,EN)	A FUNCTION CALL, SEE I.3.6.
1	NAME [E1,E2,...,EN]	A STRUCTURE ACCESS, SEE I.3.5.
1	NAME	A POINTER TO THE NAMED ITEM, SEE I.1.4.
1	LITERAL	VALUE OF THE CONVERTED LITERAL, SEE I.1.3.
3	.E .E<E1,E2>	VALUE POINTED AT BY E, POSSIBLY MODIFIED BY POSITION AND SIZE E1 AND E2. (E1 AND E2 MUST EVALUATE TO COMPILE TIME CONSTANTS.)
4	E1↑E2	E1 SHIFTED ARITHMETICALLY BY E2 BITS. SHIFT IS LEFT IF E2 IS POSITIVE, RIGHT IF NEGATIVE. E2 MUST EVALUATE TO A CONSTANT AT COMPILE TIME. (TEMPORARILY)
4	E1 ROT E2	ROTATE E1 BY E2 BITS. ROTATE LEFT IF E2 IS POSITIVE, RIGHT IF NEGATIVE. E2 MUST BE A COMPILE TIME CONSTANT.
6	-E	NEGATIVE OF E.
6	E1+E2	SUM OF E'S.
6	E1-E2	DIFFERENCE BETWEEN E1 AND E2.

[NOTE ALL ARITHMETIC IS CARRIED OUT MODULO 2(16) WITH A RESIDUE OF -2(15). ALL ARITHMETIC IS INTEGER; WHEN FLOATING POINT ARITHMETIC IS INTRODUCED IT WILL BE BY MEANS OF SPECIAL OPERATORS, VIZ., FMPR, FDVR, FNEG, FADR, FSBR.]

PRECEDENCE	EXAMPLE	SEMANTICS
7	E1 EQL E2	E1 = E2
7	E1 NEQ E2	NOT (E1 = E2)
7	E1 LSS E2	E1 < E2
7	E1 LEQ E2	NOT (E1 > E2)
7	E1 GTR E2	E1 > E2
7	E1 GEQ E2	NOT (E1 < E2)
7 7 7 7 7 7	E1 EQLU E2 E1 NEQU E2 E1 LSSU E2 E1 LEQU E2 E1 GTRU E2 E1 GEQU E2	UNSIGNED VERSIONS OF THE RELATIONAL OPERATORS, IE, TREAT E1 AND E2 AS 16 BIT UNSIGNED INTEGERS
8	NOT E	BITWISE COMPLEMENT OF E
9	E1 AND E2	BITWISE AND OF E'S
10	E1 OR E2	BITWISE INCLUSIVE OR OF E'S
11	E1 XOR E2	BITWISE EXCLUSIVE OR
11	E1 EQV E2	BITWISE EQUIVALENCE
12	E1 ← E2 E1<E3,E4>←E2 OR E1 = E2 E1<E3,E4>=E2	THE VALUE OF THIS EXPRESSION IS IDENTICAL TO THAT OF E2, BUT AS A SIDE EFFECT THIS VALUE IS STORED INTO THE PARTIAL WORD POINTED TO BY E1; WITH ASSOCIATIVE USE OF =, THE ASSIGNMENTS ARE EXECUTED FROM RIGHT TO LEFT: THUS E1 = E2 = E3 MEANS E1 = (E2 = E3).



IN GENERAL, THERE IS NO GUARANTEE REGARDING THE ORDER IN WHICH A SIMPLE EXPRESSION IS EVALUATED OTHER THAN THAT PROVIDED BY PRECEDENCE AND NESTING: THUS (R = 2; .R * (R = 3)) MAY EVALUATE TO 6 OR 9, SINCE THE OPTIMIZATION STRATEGIES MAY CAUSE THE EXPRESSION .R TO BE EVALUATED BEFORE OR AFTER (R=3), DEPENDING ON CONTEXT.

THE READER SHOULD REFER TO THE PDP-11 REFERENCE MANUAL FOR A COMPLETE DEFINITION OF THE ARITHMETIC OPERATORS UNDER VARIOUS SPECIAL INPUT VALUE CONDITIONS.

NOTE: AS OF THIS DATE (15 DEC) THE FOLLOWING OPERATORS ARE NOT YET IMPLEMENTED:

ROT	ROTATE
EQLU	
NEQU	
LSSU	UNSIGNED RELATIONALS
LEQU	
GTRU	
GEQU	

ON TARGET MACHINES WITH SPECIAL HARDWARE OPTIONS, ADDITIONAL OPERATORS MAY BE INTRODUCED CORRESPONDING TO THOSE OPERATIONS. PRIME EXAMPLES ARE:

(WITH PDP-11/45)

	FNEG E	FLOATING NEGATION
6	E1 FADR E2	FLOATING ADDITION
6	E1 FSBR E2	FLOATING SUBTRACTION
5	E1 FMPR E2	FLOATING MULTIPLICATION
5	E1 FDVR E2	FLOATING DIVISION
5	E1 FREM E2	FLOATING REMAINDER
6	FIX E	FIX A FLOATING VALUE
6	FLOAT E	FLOAT A FIXED VALUE

EXAMPLES:

```
A = .B+.C↑2;
C = .B<1,2>;
X = .B+(Y=.Z/2);
```

I.2.3.1 CONTROL EXPRESSIONS

THE CONTROLEXPRESSIONS PROVIDE SEQUENCING CONTROL OVER THE EXECUTION OF A PROGRAM; THERE ARE FIVE FORMS:

CONTROLEXPRESSION ::= CONDITIONALEXPRESSION / LOOPEXPRESSION / CHOICEEXPRESSION / ESCAPEEXPRESSION / COROUTINEEXPRESSION

I.2.3.2 CONDITIONAL EXPRESSIONS

CONDITIONALEXPRESSION ::= IF E1 THEN E2 ELSE E3

E1 IS COMPUTED AND THE RESULTING VALUE IS TESTED. IF IT IS TRUE, THEN E2 IS EVALUATED TO PROVIDE THE VALUE OF THE CONDITIONAL EXPRESSION, OTHERWISE E3 IS EVALUATED.

CONDITIONALEXPRESSION ::= IF E1 THEN E2

THIS FORM IS EQUIVALENT TO THE IF-THEN-ELSE FORM WITH 0 REPLACING E3. HOWEVER, IT DOES INTRODUCE THE "DANGLING ELSE" AMBIGUITY. THIS IS RESOLVED BY MATCHING EACH ELSE TO THE MOST RECENT UNMATCHED THEN AS THE CONDITIONAL EXPRESSION IS SCANNED FROM LEFT TO RIGHT.

EXAMPLES:

```
IF .X<8,1> THEN J = .K ELSE J=.L;

J = (IF .X<8,1> THEN .K ELSE .L);!SAME EFFECT AS PREVIOUS

IF .L THEN
    BEGIN.....END
       ELSE
    BEGIN.....END
       ;                              !BLOCKS ALLOW MULTIPLE STATEMENTS

POSITION = .POSITION + (IF .CHAR EQL #11 %TAB% THEN 8 ELSE 1);
```

I.2.3.3 LOOP EXPRESSIONS

THE VALUE OF EACH OF THE SIX LOOP EXPRESSIONS IS -1, EXCEPT WHEN AN EXITLOOP OR LEAVE IS USED, SEE 2.3.4.

LOOPEXPRESSION ::= WHILE E1 DO E2

THE E1 IS COMPUTED AND THE RESULTING VALUE IS TESTED. IF IT IS TRUE, THEN E2 IS COMPUTED AND THE COMPLETE LOOPEXPRESSION IS RECOMPUTED; IF IT IS FALSE, THEN THE LOOPEXPRESSION EVALUATION IS COMPLETE.

LOOPEXPRESSION::=UNTIL E3 DO E2

THIS FORM IS EQUIVALENT TO THE WHILE-DO FORM EXCEPT THAT E1 IS REPLACED BY NOT(E3).

LOOPEXPRESSION::=DO E2 WHILE E1

THE EXPRESSIONS E2,E1 ARE COMPUTED IN THAT SEQUENCE. THE VALUE RESULTING FROM E1 IS TESTED: IF IT IS TRUE, THEN THE COMPLETE LOOP EXPRESSION IS RECOMPUTED; IF IT IS FALSE, THEN THE LOOP-EXPRESSION EVALUATION IS COMPLETE.

LOOPEXPRESSION::=DO E2 UNTIL E3

THIS FORM IS EQUIVALENT TO THE DO-WHILE FORM EXCEPT THAT E1 IS REPLACED BY NOT(E3).

LOOPEXPRESSION::=INCR NAME FROM E1 TO E2 BY E3 DO E4

THIS IS A SIMPLIFIED FORM OF THE ALGOL 68 FOR-LOOP. THE "NAME" IS DECLARED TO BE A REGISTER OR A LOCAL FOR THE SCOPE OF THE LOOP. THE EXPRESSION E1 IS COMPUTED AND STORED IN NAME. THE EXPRESSIONS E2 AND E3 ARE COMPUTED AND STORED IN UNNAMED LOCAL MEMORY WHICH FOR EXPLANATION PURPOSES WE SHALL NAME U2 AND U3. ANY OF THE PHRASES "FROM E1" "TO E2" OR "BY E3" MAY BE OMITTED--IN WHICH CASE DEFAULT VALUES OF E1 = 0, E2 = 2↑16-1, E3 = 1 ARE SUPPLIED. THE FOLLOWING LOOPEXPRESSION IS THEN EXECUTED:

```
        BEGIN REGISTER NAME; LOCAL U2,U3; U2=E2; U3=E3;
        UNTIL .NAME GTR .U2 DO (E4; NAME=.NAME+.U3)
        END
```

THE FINAL FORM OF A LOOPEXPRESSION IS:

```
        LOOPEXPRESSION::=DECR NAME FROM E1 TO E2 BY E3 DO E4
```

THIS IS EQUIVALENT TO THE INCR-FROM-TO-BY-DO FORM EXCEPT THAT THE FINAL LOOP IS REPLACED BY

```
        BEGIN REGISTER NAME; LOCAL U2,U3; U2=E2; U3=E3;
        UNTIL .NAME LSS .U2 DO (E4; NAME=.NAME - .U3)
        END
```

IF ANY OF THE FROM, TO, OR BY PHRASES ARE OMITTED FROM A DECR EXPRESSION, DEFAULT VALUES OF E1=0, E2=-2↑15, AND E3=1 ARE SUPPLIED. NOTICE THAT IN BOTH FORMS THE END CONDITION IS TESTED BEFORE THE LOOP, HENCE THE LOOP IS POTENTIALLY EXECUTED ZERO OR MORE TIMES.

EXAMPLES:

```
LINK=.BEGINNINGOFLINKEDLIST;
WHILE..LINK NEQ 0 DO BEGIN LINK=..LINK END;!FIND LAST ITEM
                                           !ON LIST


!ADD UP FIRST N NUMBERS
 SUM=0;
 INCR J FROM 1 TO .N DO SUM=.SUM+.J;
```

I.2.3.4 ESCAPE EXPRESSIOUS

THE ESCAPE EXPRESSIOUS PERMIT CONTROL TO LEAVE ITS CURRENT ENVIRONMENT. THERE ARE THREE FORMS:

```
ESCAPEEXPRESSION::=     LEAVE LABEL WITH E  /
                        EXITLOOP E          /
                        RETURN E            /
                        LEAVE LABEL
```

ANY EXPRESSION MAY BE LABELED BY PRECEEDING IT WITH THE LABEL NAME AND A COLON. WITHIN A LABELED EXPRESSION, CONTROL MAY BE CAUSED TO LEAVE THE EXPRESSION AND YIELD THE GIVEN VALUE AS THE VALUE OF THE EXPRESSION.

THE LEAVE EXPRESSION MUST OCCUR WITHIN THE CONTROL SCOPE OF THE LABEL NAMED. THE SAME LABEL MAY ONLY BE USED ONCE WITHIN THE LEXICAL SCOPE OF ITS DEFINITION.

EXITLOOP IS A SPECIAL CASE WHICH EXITS THE SCOPE OF THE INNERMOST LOOP CONTROL SCOPE WHICH CONTAINS IT. THE VALUE BECOMES THE VALUE OF THE EXITED LOOP.

RETURN EXITS THE ROUTINE BODY AND PASSES THE GIVEN VALUE AS THE VALUE OF THE ROUTINE.

IF THE "WITH E" IS MISSING THEN "WITH 0" IS PRESUMED.

EXAMPLES:

```
!FIND INDEX OF FIRST SPACE IN LINE IMAGE OF 80 CHARACTERS
!INDEX IS -1 IF NONE FOUND

INDEX←INCR J FROM 0 TO 79 DO IF .(LINE+.J)<0,8> EQL #40
             THEN EXITLOOP .J;

!HOW TO "BRANCH" TO NEXT ITERATION IN A LOOP
INCR J FROM 1 TO 100 DO
        L2:(...
         ...
         IF .CONDITION THEN LEAVE L2;!EXIT THE COMPOUND WHICH
         ...                          !IS THE LOOP BODY,IE.,
        );                            !JUMP TO THE END OF THE
                                      !ITERATION (IF ANY).

!FIND FIRST ZERO ELEMENT OF A 2-D ARRAY
 L3:(INCR I FROM 1 TO .IMAX DO
 INCR J FROM 1 TO .JMAX DO
        IF .ARRAY[.I,.J] EQL 0 THEN (II=.I;JJ=.J; LEAVE L3);.
```

I.2.3.5 CHOICE EXPRESSIONS

```
CHOICEEXPRESSION::= CASE E OF SET EXPRESSIONSET
                                 TES
EXPRESSIONSET ::= /E/;EXPRESSIONSET/E;EXPRESSIONSET
```

THE EXPRESSION E IS EVALUATED AND USED TO SELECT THE E-TH EXPRESSION OF THE FOLLOWING SET TO EXECUTE. THE FIRST EXPRESSION IS EXECUTED IF E EQUALS 0, THE SECOND IF E EQUALS 1,ETC. THE VALUE OF THE CASE EXPRESSION IS UNDEFINED IF E IS LESS THAN ZERO OR GREATER THEN THE NUMBER OF EXPRESSIONS.

EXAMPLES:

```
!SUPPOSE TYPE(.CHAR)=0 IF .CHAR IS A NUMBER, 1 IF A LETTER,
!     2 IF IGNORABLE, EQ. SPACE, TAB, AND 3 OTHERWISE.
!COLLECT A FORTRAN STYLE IDENTIFIER, WITH COUNT BEING THE LENGTH

COUNT=0;
IF TYPE (.CHAR) EQL 1 THEN
        DO
            CASE TYPE (.CHAR) OF
                SET
                ((ID+.COUNT)<0,8>=.CHAR;COUNT=.COUNT+1);  !CASE=0
                ((ID+.COUNT)<0,8>=.CHAR;COUNT=.COUNT+1);  !CASE=1
                0;                                        !CASE=2
                EXITCONDITIONAL                           !CASE=3
                TES
        WHILE(CHAR=NEXTCHARFROMINPUT();1);
```

```
CHOICEEXPRESSION::= SELECT E OF NSET NEXPRESSIONSET
                    TESN

NEXPRESSIONSET::= / NE / NE; NEXPRESSIONSET

NE::= E:E / OTHERWISE:E/ALWAYS:E
```

THIS FORM IS SOMEWHAT SIMILAR TO THE CASE EXPRESSION EXCEPT THAT THE EXPRESSION IN THE NEXPRESSIONSET ARE NOT THOUGHT OF AS BEING SEQUENTIALLY NUMBERED--INSTEAD EACH EXPRESSION IN THE NEXPRESSIONSET IS TAGGED WITH AN "ACTIVATION" EXPRESSION. SUPPOSE WE HAVE THE FOLLOWING SELECT EXPRESSION

```
SELECT E1  OF NSET E4: E5; E6: E7; E8: E9; E10:
E11 TESN
```

THEN THE EXECUTION PROCEEDS AS FOLLOWS: FIRST E1 IS EVALUATED, THEN E4, E6, E8, AND E10 ARE EVALUATED; CORRESPONDINGLY E5 IS EVALUATED IF AND ONLY IF E4 IS EQUAL TO E1, ETC. AS WITH THE CASE EXPRESSION, THE ORDER OF EVALUATION OF THE NSET ELEMENTS IS NOT DEFINED AND THE VALUE OF THE ENTIRE EXPRESSION IS THAT OF THE LAST ONE TO BE EXECUTED AT EXECUTION TIME. THUS, THE VALUE OF THE COMPLETE SELECT EXPRESSION IS UNIQUELY DETERMINED ONLY IN THE CASE THAT THE ELIST CONTAINS PRECISELY ONE ELEMENT.

AN EXCAPE EXPRESSION IS ILLEGAL WHERE ITS EXECUTION WOULD IMPLY EXCAPE FROM AN NSET-TESN ENVIRONMENT.

IN PLACE OF ONE OF THE SELECTION EXPRESSIONS, E(4), E(6), ETC. ONE OF THE TWO RESERVED WORDS OTHERWISE OR ALWAYS MAY BE USED, E.G., "ALWAYS:E(9)". THE EXPRESSION FOLLOWING AN "OTHERWISE:" WILL BE EXECUTED JUST IN THE CASE THAT NONE OF THE PRECEDING SELECTION CRITERIA WERE SATISFIED. THE EXPRESSION FOLLOWING AN "ALWAYS:" WILL ALWAYS BE EXECUTED INDEPENDENT OF THE SELECTION CRITERIA. IN THE FOLLOWING EXAMPLE

```
Z-SELECT .X OF
         NSET
         1:E1;
         7:E2;
OTHERWISE:E3;
        36:E4;
    ALWAYS:E5;
        94:E6
        TESN
```

(1) E1 WILL BE EXECUTED IF .X=1 OR .Y=1, THEN (2) E2 WILL BE EXECUTED IF .X=7 OR .Y=7, THEN (3) E3 WILL BE EXECUTED IN THE CASE NEITHER E1 NOR E2 WAS EXECUTED, I.E., .X NOT = TO 1, .Y NOT = TO 1, .X NOT = TO 7, AND .Y NOT = TO 7, THEN (4) E4 WILL BE EXECUTED IF .X=36 OR .Y=36, THEN (5) E5 WILL ALWAYS BE EXECUTED, AND FINALLY (6) E6 WILL BE EXECUTED IF .X=94 OR.Y=94. THE VALUE ASSIGNED TO Z WILL BE THAT OF E5 UNLESS .X=94 OR .Y=94 IN WHICH CASE THE VALUE ASSIGNED TO Z WILL BE THAT OF E(6).

NOTE THAT ALTHOUGH OTHERWISE AND ALWAYS MAY BE PLACED IN ANY NSET-ELEMENT, IT MAKES NO SENSE TO USE MORE THAN ONE OTHERWISE OR TO USE AN OTHERWISE AFTER AN ALWAYS SINCE IN THESE CASES THE LATTER OTHERWISE'S CAN HAVE NO EFFECT.

EXAMPLE:

```
!SUPPOSE WE HAVE A TELETYPE INPUT ROUTINE
 SELECT .CHAR OF NSET

#15: ...          ;        !CR=> ECHO LF
#11: ...          ;        !TAB=> ECHO SPACES
     ...
         TESN
```

I.2.3.6 CO-ROUTINE EXPRESSIONS

THE ABILITY TO CREATE AND COMMUNICATE BETWEEN CO-ROUTINES IS VERY DESIRABLE. HOWEVER, CONSIDERATION IS DEFERRED UNTIL A PROPOSAL CAN BE WORKED OUT WITH THE MONITOR GROUP.

I.3.1 DECLARATIONS

ALL DECLARATIONS, EXCEPT MAP AND SWITCH, INTRODUCE NAMES EACH OF WHICH IS UNIQUE TO THE BLOCK IN WHICH THE DECLARATION APPEARS. EXCEPT WITH STRUCTURE AND MACRO DECLARATIONS, THE NAME INTRODUCED HAS A POINTER BOUND TO IT.

THE DECLARATIONS ARE:

```
DECLARATION::=ROUTINEDECLARATION/
              STRUCTUREDECLARATION/
              BINDDECLARATION/
              MACRODECLARATION/
              ALLOCATIONDECLARATION/
              MAPDECLARATION/
              LABELDECLARATION/
              UNDECLARATION/
              CALLSEQDECLARATION/
              SIGNALDECLARATION
```

BEFORE PROCEEDING WITH A DETAILED DISCUSSION OF THE DECLARATIONS WE SHALL GIVE AN INTUITIVE OVERVIEW OF THE EFFECT OF THESE DECLARATIONS.

I.3.1.1 STORAGE (AN INTRODUCTION)

A BLISS-11 PROGRAM OPERATES WITH AND ON A NUMBER OF STORAGE "SEGMENTS". A STORAGE SEGMENT CONSISTS OF A FIXED AND FINITE NUMBER OF "WORDS", EACH OF WHICH IS COMPOSED OF A FIXED AND FINITE NUMBER OF "BITS" (16 FOR THE PDP-11/20).

IN PRACTICE A SEGMENT GENERALLY CONTAINS EITHER PROGRAM OR DATA, AND IF THE LATTER, IT IS GENERALLY INTEGER NUMBERS, FLOATING POINT NUMBERS, CHARACTERS, OR POINTERS TO OTHER DATA. TO A BLISS-11 PROGRAM, HOWEVER, A SEGMENT MERELY CONTAINS A PATTERN OF BITS.

SEGMENTS ARE INTRODUCED INTO A BLISS-11 PROGRAM BY DECLARATIONS, CALLED ALLOCATION DECLARATIONS, FOR EXAMPLE:

```
GLOBAL G;
OWN X,Y [5], Z;
LOCAL P [100];
REGISTER R1, R2;
ROUTINE F(A,B) = .A↑.B;
```

EACH OF THESE DECLARATIONS INTRODUCES ONE OR MORE SEGMENTS AND BINDS THE IDENTIFIERS MENTIONED (E.G., G, X, Y, ETC.) TO THE NAME OF THE FIRST BYTE OF THE ASSOCIATED SEGMENT. (THE FUNCTION DECLARATION ALSO INITIALIZES THE SEGMENT NAMED "F" TO THE APPROPRIATE MACHINE CODE.)

THE SEGMENTS INTRODUCED BY THESE DECLARATIONS CONTAIN ONE OR MORE WORDS, WHERE THE SIZE MAY BE SPECIFIED (AS IN "LOCAL P[100]"), OR DEFAULTED TO ONE (AS IN "GLOBAL G;"). THE IDENTIFIERS INTRODUCED BY A DECLARATION ARE LEXICALLY LOCAL TO THE BLOCK IN WHICH THE DECLARATION IS MADE (THAT IS, THEY OBEY THE USUAL ALGOL SCOPE RULES) WITH ONE EXCEPTION - NAMELY, "GLOBAL" IDENTIFIERS ARE MADE AVAILABLE TO OTHER, SEPARATELY COMPILED MODULES. SEGMENTS CREATED BY OWN, GLOBAL, AND FUNCTION DECLARATIONS ARE CREATED ONLY ONCE AND ARE PRESERVED FOR THE DURATION OF THE EXECUTION OF A PROGRAM. SEGMENTS CREATED BY LOCAL REGISTER DECLARATIONS ARE CREATED AT THE TIME OF BLOCK ENTRY AND ARE PRESERVED ONLY FOR THE DURATION OF THE EXECUTION OF THAT BLOCK. REGISTER SEGMENTS DIFFER FROM LOCAL SEGMENTS ONLY IN THAT THEY ARE ALLOCATED FROM THE MACHINE'S ARRAY OF 6 GENERAL PURPOSE (FAST) REGISTERS. RE-ENTRY OF A BLOCK BEFORE IT IS EXITED (BY RECURSIVE FUNCTION CALLS, FOR EXAMPLE) BEHAVES AS IN ALGOL, THAT IS, LOCAL AND REGISTER SEGMENTS ARE DYNAMICALLY LOCAL TO EACH INCARNATION OF THE BLOCK.

THERE ARE TWO ADDITIONAL DECLARATIONS WHOSE EFFECT IS TO BIND IDENTIFIERS TO NAMES, BUT WHICH DO NOT CREATE SEGMENTS; EXAMPLES ARE:

```
EXTERNAL        S;
BIND    Y2 = Y+2, PA = P+.A;
```

AN EXTERNAL DECLARATION BINDS ONE OR MORE IDENTIFIERS TO THE NAMES REPRESENTED BY THE SAME IDENTIFIER DECLARED GLOBAL IN ANOTHER, SEPARATELY COMPILED MODULE. THE BIND DECLARATION BINDS ONE OR MORE IDENTIFIERS TO THE VALUE OF AN EXPRESSION AT BLOCK ENTRY TIME. AT LEAST POTENTIALLY THE VALUE OF THIS EXPRESSION MAY NOT BE CALCULABLE UNTIL RUN TIME - AS IN 'PA = P+.A' ABOVE.

I.3.1.2 DATA STRUCTURES (AN INTRODUCTION)

TWO PRINCIPLES WERE FOLLOWED IN THE DESIGN OF THE DATA STRUCTURE FACILITY OF BLISS:

-THE USER MUST BE ABLE TO SPECIFY THE ACCESSING ALGORITHM FOR ELEMENTS OF A STRUCTURE,
-THE REPRESENTATIONAL SPECIFICATION AND THE SPECIFICATION OF ALGORITHMS WHICH OPERATE ON THE INFORMATION REPRESENTED MUST BE SEPARATED IN SUCH A WAY THAT EITHER CAN BE MODIFIED WITHOUT AFFECTING THE OTHER.

THE DEFINITION OF A CLASS OF STRUCTURES, THAT IS, OF AN ACCESSING ALGORITHM TO BE ASSOCIATED WITH CERTAIN SPECIFIC DATA STRUCTURES, MAY BE MADE BY A DECLARATION OF SOMEWHAT THE FOLLOWING FORM:

```
STRUCTURE <NAME>[<FORMAL PARAMETER LIST>] = E
```

PARTICULAR NAMES MAY THEN BE ASSOCIATED WITH A STRUCTURE CLASS, THAT IS WITH AN ACESSING ALGOITHM, BY ANOTHER DECLARATION OF SOMEWHAT THE FORM:

```
MAP <NAME> <NAME LIST>
```

CONSIDER THE FOLLOWING EXAMPLE:

```
BEGIN
        STRUCTURE ARY2[I,J] = (.ARY2+(.I-1)*10+(.J-1));
        OWN X[100],Y[100],Z[100];
        MAP ARY2 X:Y:Z;
        .
        .
        .
        X[.A,.B] ← .Y[.B,.A];
        .
        .
        .
        END;
```

IN THIS EXAMPLE WE INTRODUCE A VERY SIMPLE STRUCTURE, ARY2, FOR TWO DIMENSIONAL (10X10) ARRAYS, DECLARE THREE SEGMENTS WITH NAMES 'X', 'Y', AND 'Z' BOUND TO THEM, AND ASSOCIATE THE STRUCTURE CLASS 'ARY2' WITH THESE NAMES. THE SYNTACTIC FORMS "X[E1,E2]" AND "Y[E3,E4]" ARE VALID WITHIN THIS BLOCK AND DENOTE EVALUATION OF THE ACCESSING ALGORITHM DEFINED BY THE ARY2-STRUCTURE DECLARATION (WITH AN APPROPRIATE SUBSTITUTION OF ACTUAL FOR FORMAL PARAMETERS).

ALTHOUGH THEY ARE NOT IMPLEMENTED IN THIS WAY, FOR PURPOSES OF EXPOSITION ONE MAY THINK OF THE STRUCTURE DECLARATION AS DEFINING A FUNCTION WITH ONE MORE FORMAL PARAMETER THAN IS EXPLICITYLY MENTIONED. FOR EXAMPLE, THE STRUCTURE DECLARATION IN THE PREVIOUS EXAMPLE:

```
STRUCTURE ARY2[I,J] = (.ARY2+(.I-1)*10+(.J-1));
```

CONCEPTUALLY IS IDENTICAL TO A FUNCTION DECLATION

```
FUNCTION ARY2(F0,F1,F2) = (.F0+(.F1-1)*10+(.F2-1));
```

THE EXPRESSIONS "X[.A,.B]" AND "Y[.B,.A]" CORRESPOND TO CALLS ON THIS FUNCTION - I.E., TO "ARY2(X,.A,.B)" AND "ARY2(Y,.B,.A)".

SINCE, IN A STRUCTURE DECALARTION, THERE IS AN IMPLICIT, UN-NAMED FORMAL PARAMETER, THE NAME OF THE STRUCTURE CLASS ITSELF IS USED TO DENOTE THIS "ZERO-TH" PARAMETER. THIS CONVENTION MAINTAINS THE POSITIONAL CORRESPONDENCE OF ACTUALS AND FORMALS. THUS, IN THE EXAMPLE ABOVE, ".ARY2" DENOTES THE VALUE OF THE NAME OF THE PARTICULAR SEGMENT BEING REFERENCED, AND 'X[.A,.B]' IS EQUIVALENT TO:

```
(X+(.A-1)*10+(.B-1))
```

THE VALUE OF THIS EXPRESSION IS A POINTER TO THE DESIGNATED ELEMENT OF THE SEGMENT NAMED BY X.

IN THE FOLLOWING EXAMPLE THE STRUCTURE FACILITY AND BIND DECLARATION HAVE BEEN USED TO ENCODE A MATRIX PRODUCT (Z(I,J) = X*Y). IN THE INNER BLOCK THE NAMES 'XR' AND 'YC' ARE BOUND TO POINTERS TO THE BASE OF A SPECIFIED ROW OF X AND COLUMN OF Y RESPECTIVELY. THESE IDENTIFIERS ARE THEN ASSOCIATED WITH STRUCTURE CLASSES WHICH ALLOW ONE-DIMENSIONAL ACCESS.

```
BEGIN
        STRUCTURE       ARY2[I,J] = (.ARY2+(.I-1)*10+(.J-1)),
                        ROW[I] = (.ROW+.I-1),
                        COL[J] = (.COL+(.J-1)*10;
        OWN             X[100],Y[100],Z[100];
        MAP             ARY2 X;Y;Z;
        .
        .
        .
        INCR I FROM 1 TO 10 DO
        BEGIN BIND XR = X[.I,1], ZR = Z[.I,1]; MAP ROW XR;ZR;
            INCR J FROM 1 TO 10 DO
            BEGIN
            REGISTER T; BIND YC=Y[1,.J]; MAP COL YC;
            T ← ;
            INCR K FROM 1 TO 10 DO T ← .T+.XR[.K]*.YC[.K];
            ZR[.J] ← .T;
            END;
        END;
        .
        .
END
```

I.3.1.3 THE ACTUAL DECLARATION SYNTAX

THE EXAMPLE DECLARATIONS IN THE PRCEDING TWO SUB-SECTIONS ARE VALID BLISS SYNTAX; HOWEVER, THEY DO NOT REFLECT THE COMPLETE POWER OF THE DECLARATIVE FACILITYIES. THE FOLLOWING SECTIONS (3.2 - 3.5) ARE DEFINITIVE PRESENTATIONS OF THE ACTUAL SYNTAX AND SEMANTICS OF THESE DECLARATIONS. THE ACTUAL DECLARATIONS PRESENTED IN THE FOLLOWING SECTIONS DIFFER FROM THE EXAMPLES GIVEN PREVIOUSLY IN THAT THEY ADMIT GREATER INTERACTION BETWEEN THE ALLOCATION DECLARATIONS AND STRUCTURE DECLARATIONS.

I.3.2 MEMORY ALLOCATION

THERE ARE FIVE BASIC FORMS OF ALLOCATION DECLARATION:

```
ALLOCATION DECLARATION ::= ALLOCATETYPESIZE MSIDLIST
ALLOCATETYPESIZE::= ALLOCATESIZE ALLOCATETYPE/ALLOCATESIZE ALLOCATETYPE CALLT
ALLOCATSIZE::= /BYTE/WORD
ALLOCATETYPE ::= GLOBAL/REGISTER/OWN/LOCAL/EXTERNAL
MSIDLIST ::= MSIDELEMENT/MSIDELEMENT,MSIDLIST
MSIDELEMENT ::= STRUCTURE SIZEDCHUNKS
STRUCTURE ::= / STRUCTURENAME
SIZEDCHUNKS ::= SIZEDCHUNK/SIZEDCHUNK:  SIZEDCHUNKS
SIZECHUNK ::= IDCHUNK/IDCHUNK [ELIST]
IDCHUNK ::= NAME/NAME:  IDCHUNK
```

AS WITH MOST OTHER DECLARATIONS, THE ALLOCATION DECLARATIONS INTRODUCE NAMES WHOSE SCOPE IS THE BLOCK IN WHICH THE DECLARATIONS OCCUR. REGISTER AND LOCAL DECLARATIONS CAUSE ALLOCATION OF STORAGE AT EACH BLOCK ENTRY (INCLUDING RECURSIVE AND QUASI-PARALLEL ONES), AND CORRESPONDING DE-ALLOCATION ON BLOCK EXIT. STORAGE FOR OWN AND GLOBAL DECLARATIONS IS MADE ONCE (BEFORE EXECUTION BEGINS) AND REMAINS ALLOCATED DURING THE ENTIRE EXECUTION OF THE PROGRAM. EXTERNAL DECLARATIONS DO NOT ALLOCATE STORAGE, BUT CAUSE A LINKAGE TO BE ESTABLISHED TO STORAGE DECLARED WITH THE SAME NAME IN A GOLBAL DECLARATION OF ANOTHER MODULE. SPACE FOR ALLOCATION IS TAKEN FROM CORE FOR LOCAL, OWN, AND GLOBAL DECLARATIONS, AND FROM THE MACHINE'S HIGH SPEED REGISTERS FOR REGISTER DECLARATIONS. ALLOCATESIZE SPECIFIES WHAT UNIT SIZE IS TO BE ALLOCATED. IF THIS IS NOT PRESENT THEN WORD IS ASSUMED.

THE INITIAL CONTENTS OF ALLOCATED MEMORY IS NOT DEFINED AND SHOULD NOT BE PRESUMED.

EACH MSIDELEMENT DEFINES A SET OF IDENTIFIERS AND SIMULTANEOUSLY MAPS THESE IDENTIFIERS ONTO A SPECIFIED STRUCTURE. (IF THE STRUCTURE PART IS EMPTY, THE DEFAULT STRUCTURE 'VECTOR' IS ASSUMED, SEE SECTION 3.5). EACH SIZED CHUNK ALLOWS, BY INTERACTION WITH THE ASSOCIATED STRUCTURE OF THE MSIDELEMENT, SPECIFICATION OF THE SIZE OF THE SEGMENT TO BE ALLOCATED - AND THE VALUES OF THE "UNDOTTED STRUCTURE FORMALS" TO BE USED IN ACCESSING AN INSTANCE OF THE STRUCTURE (AGAIN, SEE I.3.5).

I.3.3 MAP DECLARATION

MAP DECLARATION ::= MAP MSIDLIST/MAP CALLTYPE MSIDLIST

THE MAP DECLARATION IS SYNTACTICALLY AND SEMANTICALLY SIMILAR TO AN ALLOCATION DECLARATION EXCEPT THAT NO NEW STORAGE OR IDENTIFIERS ARE INTRODUCED. THE PURPOSE OF THE MAP DECLARATION IS TO PERMIT RE-DEFINITION OF THE STRUCTURE AND ELIST INFORMATION ASSOCIATED WITH AN IDENTIFIER (OR SET OF IDENTIFIERS) FOR THE SCOPE OF THE BLOCK IN WHICH THE MAP DECLARATION OCCURS.

I.3.4 BIND DECLARATIONS

```
BIND DECLARATION ::= BIND EQUIVALENCELIST
EQUIVALENCELIST ::= EQUIVALENCE/EQUIVALENCE, EQUIVALENCELIST
EQUIVALENCE ::= MSIDELEMENT = E
```

A BIND DECLARATION INTRODUCES A NEW SET OF NAMES WHOSE SCOPE IS THE BLOCK IN WHICH THE BIND DECLARATION OCCURS, AND BINDS THE VALUE OF THESE NAMES TO THE VALUE OF THE ASSOCIATED EXPRESSIONS AT THE TIME THAT THE BLOCK IS ENTERED. NOTE THAT THESE EXPRESSIONS NEED NOT EVALUATE AT COMPILE TIME.

I.3.5 STRUCTURES

STRUCTURE DECLARATION ::= STRUCTURE NAME STRUCTUREFORMALLIST
= STRUCTURESIZE E
STURCTUREFORMALLIST ::= / [NAMELIST]
STRUCTURESIZE ::= / [E]

STRUCTURE DECLARATIONS SERVE TO DEFINE A CLASS OF DATA STRUCTURES BY DEFINING AN EXPLICIT "ACCESS ALGOITHM", E, TO BE USED IN ACCESSING ELEMENTS OF THAT STRUCTURE. THE CLASS OF STRUCTURES INTRODUCED BY SUCH A DECLARATION IS GIVEN A NAME WHICH MAY BE USED AS THE STRUCTURE NAME IN AN ALLOCATION DECLARATION OR MAP DECLARATION.

THE ACCESSING ALGORITHM, E, MAY NOT CONTAIN ANY DECLARATIONS.

THE NAMES IN THE STRUCTURE FORMAL LIST ARE FORMAL PARAMETER IDENTIFIERS WHICH ARE USED IN TWO DISTINCT WAYS:

1. "DOTTED" OCCURRENCES OF THE FORMAL NAMES POSITIONALLY CORRELATE WITH THE VALUES OF ELIST ELEMENTS AT THE SITE OF A STRUCTURE ACCESS. (RECALL THAT A STRUCTURE ACCESS IS SYNTACTICALLY NAME [ELIST].) THESE ARE REFERRED TO AS "ACCESS FORMALS" AND "ACCESS ACTUALS" RESPECTIVELY.

2. "UNDOTTED" OCCURRENCES OF THE FORMAL NAMES POSITIONALLY CORRELATE WITH THE VALUES OF THE ELIST ELEMENTS AT THE SITE OF THE DECLARATION WHICH ASSOCIATED THE VARIABLE NAME WITH THE STRUCTURE CLASS. THESE ARE REFERRED TO AS "INCARNATION FORMALS" AND "INCARANATION ACTUALS" RESPECTIVELY.

IN ADDITION TO THE EXPLICIT FORMAL NAMES, THE STRUCTURE NAME, IN "DOTTED" FORM, IS USED AS AN ACCESS FORMAL TO DENOTE THE NAME OF THE SPECIFIC SEGMENT BEING ACCESSED (THAT IS, TO DENOTE THE POINTER TO THE BASE OF THE SEGMENT).

IF PRESENT, THE STRUCTURE SIZE, I.E., [E], IS USED TO CALCULATE (FROM THE INCARNATION ACTUALS) THE SIZE OF THE SEGMENT TO BE ALLOCTED BY AN ALLOCATION DECLARATION. AFTER SUBSTITUTION OF INCARNATION ACTUALS, THIS EXPRESSION MUST EVALUATE TO A CONSTANT AT COMPILE TIME.

THE SIMPLE EXAMPLE OF A TWO-DIMENSIONAL ARRAY GIVEN IN SECTION 3.1.2 MIGHT NOW BE WRITTEN:

```
BEGIN
        STRUCTURE ARY2[I,J] = [I*J](.ARY2+(.I-1)*J+(.J-1));
        OWN ARY2 X:Y:Z[10,10];
        .
        .
        .
        X[.A,.B] ← .Y[.B,.A];
        .
        .
        .
END;
```

THE DEFAULT STRUCTURE VECTOR, MENTIONED IN SECTION 3.2 IS DEFINED BY

```
        STRUCTURE VECTOR [I] = [I](.VECTOR + .I)<0,16>;
```

IF DEFAULTED, THE SIZE PART OF A STRUCTURE DECLARATION ([I*J] IN THE ABOVE EXAMPLE) IS DEFAULTED TO THE PRODUCT OF THE INCARNATION ACTUALS (X:Y:Z[10,10] IN THE ABOVE EXAMPLE).

I.3.6 ROUTINES

```
ROUTINEDECLARATION::= ROUTINE NAME(NAMELIST) = E /
                      ROUTINE NAME = E
```

THE ROUTINE DECLARATION DEFINES THE NAME TO BE THAT OF A POTENTIALLY RECURSIVE AND RE-ENTRANT FUNCTION WHOSE VALUE IS THE EXPRESSION E. THE SYNTAX OF A NORMAL SUBROUTINE-LIKE FUNCTION CALL IS

```
P1::= P1 (ELIST) / P1 ()
ELIST::= E / ELIST, E
```

WHERE P1 IS a PRIMARY EXPRESSION. CLEARLY, P1 MUST EVALUATE TO A NAME WHICH HAS BEEN DECLARED AS A ROUTINE EITHER AT COMPILE TIME OR AT RUN TIME. THE NAMES IN THE NAME-LIST OF THE DECLARATION DEFINE (LEXICALLY LOCAL) NAMES OF FORMAL PARAMETERS WHOSE ACTUAL VALUES ON EACH INCARNATION ARE DETERMINED BY THE ELIST AT THE CALL SITE. ALL PARAMETERS ARE IMPLICITLY ALGOL "CALL-BY-VALUE"; BUT NOTICE THAT CALL-BY-REFERENCE IS ACHIEVED BY SIMPLY PRESENTING POINTER VALUES AT THE CALL SITE. PARENTHESES ARE REQUIRED AT THE CALL SITE EVEN FOR A ROUTINE WITH NO FORMAL PARAMETERS SINCE THE NAME ON ITS OWN IS SIMPLY A POINTER TO THE ROUTINE.

```
DECLARATION::= GLOBAL ROUTINE NAME (NAMELIST) = E/
               GLOBAL ROUTINE NAME = E
```

A ROUTINE NAME IS LIKE AN OWN NAME IN THAT ITS SCOPE IS LIMITED TO THE BLOCK IN WHICH IT IS DECLARED AND ITS VALUE IS ALREADY INITIALIZED AT BLOCK ENTRY. THE PREFIX GLOBAL CHANGES THE SCOPE OF THE ROUTINE TO THAT OF THE OUTER BLOCK OF THE PROGRAM ENVELOPING ALL THE MODULES. NOTE THAT THIS INHIBITS A GLOBAL ROUTINE FROM ACCESS TO REGISTER NAMES DECLARED OUTSIDE IT.

```
DECLARATION::= EXTERNAL NAMEPARLIST /
               FORWARD NAMEPARLIST

NAMEPARLIST::= NAMEPAR / NAMEPARLIST, NAMEPAR

NAMEPAR::=     NAME (E) / NAME
```

EXTERNAL AND FORWARD EACH TELL THE COMPILER HOW MANY PARAMETERS, GIVEN BY E*, ARE EXPECTED BY AN UNDECLARED ROUTINE NAME. FORWARD IS FOR ROUTINES DECLARED LATER IN THE CURRENT BLOCK AND EXTERNAL IS FOR ROUTINES FROM ANOTHER MODULE. THE COMPILER PERMITS THE NUMBER OF ACTUAL PARAMETERS IN A ROUTINE CALL TO BE GREATER THAN, EQUAL TO, OR LESS THAN TO THE NUMBER OF FORMALS DECLARED. (SEE SECTION IV.1.2 FOR AN EXPLANATION OF HOW ARGUMENTS ARE PASSED ON A ROUTINE CALL. IT WILL INDICATE THE RESULT WHICH OCCURS WHEN THE NUMBER OF ARGUMENTS PASSED IS NOT THE SAME AS THE NUMBER OF FORMAL PARAMETERS IN THE ROUTINE DECLARATION.) A DEFAULT VALUE OF ZERO IS SUPPLIED IF "(E)" IS MISSING.

---

*CLEARLY E MUST EVALUATE TO A CONSTANT AT COMPILE TIME.

I.3.7 LABEL DECLARATION

LABELDECLARATION::= LABEL LABELLIST

LABELLIST::= NAME/NAME, LABELLIST

LABELS ARE USED SOLELY IN CONJUNCTION WITH THE ESCAPE EXPRESSIONS. EACH NAME TO BE USED AS A LABEL MUST BE SO DECLARED AT THE HEAD OF THE BLOCK CONTAINING THAT USAGE.

I.3.8 MACROS

IN ORDER TO FACILITATE PROGRAM READABILITY AND MODIFIABILITY, A MACRO SYSTEM IS EMBEDDED IN BLISS. THE SYSTEM ALLOWS NESTED MACRO DEFINITION AS WELL AS ITERATIVE AND RECURSIVE FORMS OF EVALUATION. SYNTAX FOR MACRO DECLARATION:*

```
DECLARATION          ::=MACRO DEFINITIONLIST
DEFINITIONLIST       ::=DEFINITION /DEFINITIONLIST,DEFINITION
DEFINITION           ::=NAME FIXEDPARMS ITERATEDPARMS=STRING $
FIXEDPARMS           ::=/(NAMELIST)
ITERATEDPARMS        ::=/[]/[NAMELIST]
```

THE ESSENTIAL FUNCTION OF THE MACRO SYSTEM IS TO REPLACE THE THE MACRO NAME AND ITS ACTUAL-PARAMETER LIST (WHEREEVER THE NAME OCCURS WITHIN ITS SCOPE IN THE PROGRAM) BY ITS BODY, WITH ACTUAL-PARAMETERS SUBSTITUTED FOR FORMALS. THE BODY IS CONSIDERED TO BE A STRING OF "ATOMS"--NAMES, LITERALS AND DELIMITERS--AND IS THEREFORE INDEPENDENT OF EDITING SYMBOLS--BLANKS, CR,LF,AND BLISS COMMENTS --ONCE THE ATOMS ARE DETERMINED AT MACRO DECLARATION TIME.

THE FORMAT OF THE MACRO CALL IS SIMPLY THE MACRO-NAME FOLLOWED BY THE BRACKETED ACTUAL-PARAMETER LIST. THE BRACKETS MUST BE ONE OF THE PAIRS: (), [], <>, AND THE ACTUAL-PARAMETERS MUST BE SEPARATED BY COMMAS. THE ACTUAL PARAMETERS THEMSELVES MAY BE ARBITRARY STRINGS OF ATOMS; HOWEVER, OCCURRENCES OF THE BRACKETS: (), [], <>, AND MUST BE NESTED.* ALL MACROS IN ACTUAL-PARAMETER LISTS ARE EXPANDED BEFORE FORMAL/ACTUAL BINDING.*

```
LET:  MACRO  SUBSTITUTE=BODY(1) $,
             SIMPLE(F(1),...F(N))=BODY(2) $,
             PASS[]=BODY(3) $,
             RECURSIVE(F(1),...F(N))[]=BODY(4) $,
             ITERATED1[(1),...I(N)]=BODY(5) $,
             ITERATED2(F(1),...,F(N))[I(1),...,I(M)]=BODY(6) $;
```

REPRESENT THE CANNONICAL FORMS OF MACRO DEFINITIONS PERMITTED IN BLISS. LET "MNAME (A(1),...,A(K))" REPRESENT A TYPICAL MACRO CALL, WHERE THE A(I) ARE ACTUALS. THE REPLACEMENT ALGORITHM VARIES ACCORDING TO THE FORM OF THE DECLARATION:

1. SUBSTITUTE: THE BODY REPLACES THE MACRO NAME;

2. SIMPLE: OCCURRENCES OF F(I) IN BODY(2) ARE REPLACED BY A(I); BODY(2) IS SUBSTITUTED FOR THE CALL. UNSPECIFIED ACTUALS DEFAULT TO THE EMPTY EXTRA ACTUALS ARE IGNORED BUT A WARNING MESSAGE IS ISSUED.

****

* UNLESS QUOTED--SEE "SPECIAL FUNCTIONS".

3. PASS: THE BODY REPLACES THE MACRO-NAME (AND PARAMETER LIST) ONLY IF AN ACTUAL-PARAMETER LIST IS SPECIFIED; OTHERWISE, THE MACRO IS REPLACED BY THE EMPTY ATOM. IF AN ACTUAL-PARAMETER LIST IS SPECIFIED, THE SPECIAL FUNCTION $REMAINING MAY BE USED IN BODY(3) TO STAND FOR ACTUAL-PARAMETER LIST WITH OUTER BRACKETS REMOVED AND THE "DEFAULT SEPARATOR" REPLACING THE COMMAS.*

4. RECURSIVE: IF A(N) IS NOT SPECIFIED, THE EMPTY ATOM REPLACES THE CALL. OTHERWISE, A(I) IS BOUND TO F(I) (I=1 TO N). THE REMAINING ACTUALS--A(N)+1,...,A(K)--DEFINE THE SPECIAL FUNCTION $REMAINING MENTIONED ABOVE, WHICH IS AVAILABLE FOR USE IN BODY(4). NOTE: A RECURSIVE CALL WITHIN THE BODY IS PERMITTED, NOT REQUIRED.**

5. ITERATED1: IF A(N) IS NOT SPECIFIED, THE EMPTY ATOM REPLACES THE CALL. OTHERWISE, A(I) IS BOUND TO I(I) (I=1 TO N), AND $REMAINING IS DEFINED AS IN THE RECURSIVE FORM. THE BODY IS OUTPUT. IF A2N IS NOT SPECIFIED, THE EXPANSION TERINATES. OTHERWISE, THE "DEFAULT SEPARATOR"* IS OUTPUT, I(I) TAKE ON THE STRINGS A (2I), AND THE BODY IS OUTPUT. THE PROCESS CONTINUES ITERATIVELY UNTIL THE ACTUAL-PARAMETER LIST IS EXHAUSTED (A(JN) IS NOT SPECIFIED, FOR SONE J), PRODUCING INSTANCES OF BODY, SEPARATOR, BODY... SEPARATOR, BODY.

6. ITERATED2: THE FIXED FORMALS ARE BOUND TO THE FIRST N ACTUAL-PARAMETERS--A(I) TO F(I). THE EXPANSION THEN BEHAVES AS IF A MACRO SIMILAR TO ITERATED1 WERE DEFINED; I.E., THE ITERATIVE EVALUATION IS PERFORMED WITH BINDINGS I(I) TO A(N)+I+JM (ON THE JTH ITERATION).

DEFAULT SEPARATOR:

THE SEPARATOR GENERATED IS ALWAYS A FUNCTION OF THE BLISS CONTEXT WHICH PRECEDES THE MACRO CALL. IF SOME CASES, BRACKETS ARE AUTOMATICALLY GEERATED AROUND THE REPLACED MACRO CALL.

PRECEDING CONTEXT	GENERATED SEPARATOR	GENERATED BRACKETS
SEPARATOR: , OR ;	, OR ;	
LEFT BRACKETS:		
BEGIN OR (IN A BLOCK		
OR COMPOUND	;	
SET OR NSET	;	
([<	,	
DECLARATORS:		
ALLOCATING, MACRO, MAP		
EXTERNAL AND BIND	,	
ROUTINE, GLOBAL ROUTINE,	,	
STRUCTURE		

****
*SEE "DEFAULT SEPARATOR" ABOVE.
**EXTENSIONS TO RECURSION MAY BE EXPECTED.

PRECEDING CONTEXT	GENERATED SEPARATOR	GENERATED BRACKETS
STRUCTURE NAME IN A MAPPING DECLARATION	:	
PLIT	,	()
OF AFTER CASE	;	SET TES
OF AFTER SELECT	;	NSET TESN
EXPRESSION OR NAME	,	()
OPERATOR	THE SAME OPERATOR	

SPECIAL FUNCTIONS:

1. $REMAINING: IN ANY MACRO OF THE FORMS BODY(3)-BODY(6) ABOVE, THE ACTUAL PARAMETERS NOT YET BOUND IN A MACRO EXPANSION. THE BRACKETS ARE REMOVED FROM THE PARAMETER LIST; THE SEPARATOR BETWEEN THE PARAMETERS IS DETERMINED FROM THE CONTEXT OF THE USE OF THE FUNCTION.

2. $LENGTH: THE NUMBER OF ACTUALS PASSED IN THE CURRENT MACRO CALL.

3. $COUNT: THE DEPTH OF RECURSION OF A RECURSIVE MACRO (1ST CALL IS DEPTH 0) OR THE (0-ORIGIN) ITERATION COUNT OF AN ITERATIVE MACRO (THE NUMBER OF DEFAULT SEPARATORS WHICH HAVE BEEN PRODUCED).

4. $QUOTE: THE ATOM FOLLOWING THE $QUOTE LOSES ITS NORMAL MEANING IN THE CURRENT CONTEXT:

   A. MACRO BODY DEFINITION: MUST QUOTE $, $QUOTE, AND $UNQUOTE IF THEIR USE AT MACRO DEFINITION TIME IS NOT DESIRED;
   B. MACRO ACTUAL-PARAMETER LIST EVALUATION: MUST QUOTE MACROS EXPANSION IS TO BE SUPPRESSED UNTIL AFTER ACTUAL SUBSTITUTION INTO THE BODY AND BRACKETS WHICH ARE NOT PROPERLY NESTED.

5. $UNQUOTE: THE ATOM FOLLOWING THE $UNQUOTE IS "EVALUATED" ONE LEVEL IN A MACRO BODY SPECIFICATION. THE FIRST LEVEL OF EVALUATION CAUSES NAMES TO BECOME "LEXEMES"--E.G. A LOCAL VARIABLE--; THE SECOND CAUSES MACROS TO BE EVALUATED.

6. $STRING: A PARAMETER LIST MUST BE PASSED. THE STRING VALUES OF THE LIST ARE CONCATENATED TO FORM A STRING (A SINGLE ATOM). NAMES HAVE THEMSELVES AS STRING VALUES; NUMBERS HAVE THEIR ASCII EQUIVALENT.
7. $NAME: A PARAMETER LIST MUST BE PASSED. THE STRING VALUES OF IS DETERMINED AS FOR $STRING. THE RESULTING STRING IS USED AS A NAME (IDENTIFIER)--ITS FORMAT MUST BE ACCEPTABLE TO THE LOADER (NOT NECESSARILY THE COMPILER).

EXAMPLES:

1.
```
MACRO   HBYTE=8,8$,      !POSITION AND SIZE OF HIGH-ORDER BYTE
        LBYTE=0,8$,      !POSITION AND SIZE OF LOW-ORDER BYTE
        TAB=OUTPUT(#14)$; !CALL OUTPUT ROUTINE TO SEND A TAB

        A<LBYTE>=.B<HBYTE> !MOVE A BYTE
```

2.
```
MACRO   OUTERR(NUM,MSG)=
        OUTSTRING(PLIT($STRING('ERR',NUM,':',MSG))$;

%USE THIS MACRO TO CALL AN OUTPUT ROUTINE WITH A POINTER
TO AN ERROR MESSAGE AS THE ARGUMENT %

        OUTERR(4,'INVALID DATA') IS EQUIVALENT TO

        OUTSTRING(PLIT('ERR4: INVALID DATA'))
```

3.
```
MACRO   COND(BOOLMEXP)[]=
        IF BOOL THEN EXP EL($REMAINING) COND ($REMAINING)$;
MACRO   EL[]=ELSE$;  !GENERATE ELSE ONLY IF PARAMETER IS NON-EMPTY

% CALL COND WITH PAIRS OF BOOLEAN EXPRESSIONS AND EXPRESSIONS.
THE RESULT AT EXECUTION TIME IS THE BOOLEAN EXPRESSIONS
ARE EVALUSTED UNTIL ONE IS TRUE THEN THE CORRESPONDING
EXPRESSION IS EXECUTED%

COND (.B,A=.X,.X OR .Y, FCT(.M))
  GENERATES

IF .B THEN A=.X
        ELSE IF .X OR .Y THEN FCT (.M)
```

4.
```
MACRO FLAGS (FLAGWORD)[FLAGNAME]=
        BIND FLAGNAME=FLAGWORD<$COUNT,1>$;
% THIS BINDS VARIOUS FLAG NAMES TO INDIVIDUAL BITS
  IN THE SPECIFIED WORD %

FLAGS(X,LIST,ERR,OPT,PIC)
GENERATES

        BIND  LIST=X<0,1>;
        BIND  LRR=X<1,1>;
        BIND  OPT=X<2,1>;
        BIND  PIC=X<3,1>;
```

5.
```
MACRO  REV(X)[]=REV($REMAINING)  CC($REMAINING) X$;
        CC[]=,$;  !GERERATE COMMA ONLY IF PARAMETER IS NON-EMPTY
%REVERSES THE PARAMETER LIST %

        REV (A,B,C,D)  GENERATES D,C,B,A
```

6.
```
MACRO  INITLOCALARRAY (NAME)[]=
        LOCAL NAME[$LENGTH-1];
        LOAD (NAME,$REMAINING)$;
```

```
MACRO    LOAD(BASE) [VALUE]=
             BASE [$COUNT] =VALUE$;

         INITLOCALARRAY (A,4,3,7,0,1)
     GENERATES
         LOCAL A[5];
         A[0]=4;
         A[1]=3;
         A[2]=7;
         A[3]=0;
         A[4]=1
```

I.3.9 UNDECLARE DECLARATION

DECLARATION::=UNDECLARE NAMELIST

THE IDENTIFIERS ARE IN THE NAMELIST BECOME UNDEFINED WITHIN THE SCOPE OF THE DECLARATION.

I.3.10 CALLING SEQUENCE DECLARATIONS

<CALLSEQUENCE DECLARATION> ::=

LINKAGE <IDENTIFIER><CALL TYPE>(<CALL ARGLIST>)

<CALL TYPE> ::= BLISS/EMT/TRAP
<CALL ARGLIST> ::=<CALL ARGTYPE>
<CALLARGLIST>,<CALLARGTYPE>
<CALLARGTYPE> ::=STACK /
REGISTER <INTERGER>

THE LINKAGE DECLARATION SERVES TO DEFINE THE FOLLOWING IDENTIFIER AS THE NAME OF A PARTICULAR TYPE OF CALLING SEQUENCE. IN ADDITION TO THOSE DEFINABLE BY THE USER THERE ARE TWO LINKAGE TYPES, FORTRAN AND INTERRUPT, WHICH ARE PREDEFINED.

(FORTRAN, INTERRUPT AND ANY IDENTIFIES DECLARED AS ABOVE ARE CALLED LINKAGE KEYWORDS.)

I.3.10.1 FORTRAN LINKAGE

IN BRIEF, THE FORTRAN LINKAGE IS COMPATIBLE WITH THE PDP-11/20 FORTRAN IV SO THAT FORTRAN COMPILED ROUTINES MAY CALL BLISS COMPILED ROUTINES AND VICE-VERSA. NOTE THAT FORTRAN USES A CALL-BY-REFERENCE METHOD OF PARAMETER PASSING WHILE BLISS USES CALL-BY-VALUE. THE BLISS CODER MUST EXPLICITLY ACCOMODATE THE FORTRAN COMVENTION (SEE SECTION FOR FURTHER DISCUSSION.)

(NOTE THAT THE FORTRAN LINKAGE IS INHERENTLY NON-REENTRANT SINCE FORMAL PARAMETERS ARE PASSED IN A MANNER THAT IS EFFECTIVELY LIKE AN OWN ARRAY RATHER THAN A LOCAL ARRAY.)

I.3.10.2 INTERRUPT LINKAGE

THE INTERRUPT LINKAGE SPECIFICATION IS APPROPRIATE ONLY TO ROUTINE DECLARATIONS. IT DECLARES THAT THE ROUTINE WILL BE ENTERED AS A RESULT OF AN INTERRUPT OR TRAP. NOTE THAT THIS DECLARATION ONLY COMPILES A ROUTINE FOR SERVICING INTERRUPTS. THE USER MUST SEPARATELY ARRANGE TO ESTABLISH THE ROUTINE NAME AND DESIRED STATUS WORD IN THE APPROPRIATE INTERRUPT VECTOR.

AN INTERRUPT ROUTINE MAY NOT HAVE ANY EXPLICITLY DECLARED FORMAL PARAMETERS. HOWEVER IT HAS TWO IMPLICITLY DECLARED FORMAL PARAMETERS, OLDPC AND OLDPS, WHICH MAY BE USED TO ACCESS THE PROGRAM COUNTER AND PROCESSOR STATUS SAVED ON THE STACK BY THE INTERRUPT HARDWARE.

I.3.10.3 BLISS LINKAGE

THE BLISS LINKAGE ALLOWS THE USER TO SPECIFY THAT CERTAIN VALUES WILL BE PASSED TO A SUBROUTINE ON THE EXECUTION STACK WHICL OTHERS MAY BE PASSED IN THE HARDWARE REGISTERS. IN THE DECLARATION, THE KEYWORDS STACK OR REGISTER MAY BE USED TO DESIGNATE THE METHOD OF ARGUMENT TRANSMISSION FOR THE CORRESPONDING ARGUMENT. FOR EXAMPLE,

```
LINKAGE  SST=BLISS(STACK,REGISTER 2)
```

SPECIFIES THAT THE FIRST ARGUMENT GOES ON THE STACK AND THE SECOND IS PASSED IN REGISTER 2. AN ABSOLUTE INTEGER IN THE RANGE 0 TO 5 INCLUSIVE MAY BE USED FOR THE REGISTER NUMBER.

ANY ARGUMENTS IN ADDITION TO THOSE DEFINED IN THE CALL WILL BE AUTOMATICALLY PASSED ON THE STACK. THE DEFAULT LINKAGE FOR ALL ROUTINE CALLS AND DECLARATIONS IS EQUIVALENT TO THE SPECIFICATION

```
LINKAGE  BLISS11=BLISS();
```

THAT IS, ALL ARGUMENTS ARE NORMALLY PASSED ON THE STACK.

I.3.10.4 EMT AND TRAP LINKAGE

[EMT AND TRAP CALLING LINKAGES ARE NOT CURRENTLY DEFINABLE. EMT AND TRAP SPECIAL FUNCTIONS ARE AVAILABLE AS DESCRIBED IN SECTION II.

THE GOAL IS TO BE ABLE TO CONVENIENTLY INTERFACE TO DOS,RSX, OR OTHER OPERATING SYSTEM TRAP HANDLERS. THE VARIETY OF RESPONCES TO EMT'S BY DOS, FOR EXAMPLE, HAS POSED CERTAIN PROBLEMS IN MEETING THIS OBJECTIVE. AS AN INTERIM MEASURE, A SET OF SYSTEM MACROS WILL BE PROVIDED FOR INVOKING MONITOR SERVICES. THESE ARE OUTSIDE OF THE SCOPE OF THIS DOCUMENT]

I.3.11 LINKAGE DECLARATIONS

ONCE AN IDENTIFIER HAS BEEN DEFINED TO BE A LINKAGE DECLARATION KEYWORD, IT MAY BE USED IN THE FOLLOWING WAYS WITHIN THE SCOPE OF ITS DECLARATION.

I.3.11.1 ROUTINE TYPING

PRECEEDING THE ROUTINE KEYWORD IN A ROUTINE DECLARATION. IN THIS CONTEXT THE NAMED LINKAGE TYPE DESIGNATES THE CONVENTIONS THAT WILL BE USED TO CALL THE ROUTINE BEING DECLARED, AND HENCE GOVERNS THE ENTRY AND EXIT CONVENTIONS NEEDED TO COMPILE THE ROUTINE. FOR EXAMPLE:

```
BEGIN
    ...
    FORTRAN ROUTINE MAX(A,B)=...
    ...
END
```

I.3.11.2 LINKAGE ATTRIBUTE FOR IDENTIFIERS.

FOLLOWING ONE OF THE KEYWORDS EXTERNAL, GLOBAL, OWN, OR MAP AND FOLLOWED BY A MSIDLIST. THE EFFECT IS TO SPECIFY THAT WHEN AN IDENTIFIER SO DECLARED IS USED SYNTACTLY AS THE NAME OF A ROUTINE CALL, THEN THE DECLARED CALLING SEQUENCE WILL BE USED TO EFFECT THAT CALL.

EXAMPLE:

```
FORWARD FORTRAN A,B;
EXTERNAL INTERRUPT LPT,DSK,B72;
BYTE OWN FORTRAN BYTEVECTOR F[100];
```

I.3.11.3 "LOCAL" CALL SEQUENCE DESIGNATION

WHEN THE LINKAGE NAME IS ITSELF USED SYNTACTICALLY AS A ROUTINE NAME IN A ROUTINE CALL EXPRESSION, THE THE FIRST ARGUMENT IS EVALUATED AT RUN-TIME AND INTERPRETED TO DESIGNATE THE ROUTINE ENTRY POINT TO BE CALLED. THE REMAINING ARGUMENTS ARE PASSED TO THE MANED ROUTINE IN THE MANNER REQUIRED BY THE LINKAGE DEFINITON.

EXAMPLE:

```
FORTRAN(.(TRANSFERVECTOR+.I),PAR1,PAR2)
```

II. SPECIAL LANGUAGE FEATURES

THE PREVIOUS CHAPTER DESCRIBES THE BASIC FEATURES OF THE BLISS-11 LANGUAGE. IN THIS CHAPTER WE DESCRIBE ADDITIONAL FEATURES WHICH ARE HIGHLY MACHINE AND IMPLEMENTATION DEPENDENT.

II.1.1 SPECIAL FUNCTIONS

A NUMBER OF FEATURES HAVE BEEN ADDED TO THE BASIC BLISS-11 LANGUAGE WHICH ALLOW GREATER ACCESS TO THE PDP-11 HARDWARE FEATURES. THESE FEATURES HAVE THE SYNTACTIC FORM OF FUNCTION CALLS AND ARE THUS REFERRED TO AS "SPECIAL FUNCTIONS". CODE FOR SPECIAL FUNCTIONS IS ALWAYS GENERATED IN LINE.

II.1.1.1 TRAP AND EMT SPECIAL FUNCTIONS - NOT YET IMPLEMENTED

```
<TRAP/EMT CALL> ::= EMT(<INTEGER><REST OF ARGS)/
                    TRAP(<INTEGER><REST OF ARGS)
<REST OF ARGS>  ::= /
                    E/
                    E,<REST OF ARGS>
```

AN EMT OR TRAP SPECIAL FUNCTION IS SYNTACTICALLY A FUNCTION CALL WITH AT LEAST ON PARAMETER. THE FIRST (AND POSSIBLY ONLY) PARAMETER MUST EVALUATE AT COMPILE-TIME TO A VALUE IN THE RANGE 0 TO 255 INCLUSIVE. THIS VALUE IS INCORPORATED INTO THE LOW BYTE OF THE EMT/TRAP INSTRUCTION ITSELF. THE REMAINING ARGUMENTS, (IF ANY) ARE EVALUATED AND PUSHED ONTO THE STACK PRIOR TO THE EMT/TRAP INSTRUCTION. IT IS ASSUMED THAT THE ARGUMENTS ARE REMOVED FROM THE STACK FOLLOWING RETURN FROM THE EMT/TRAP.

II.1.1.2 STACKADJUST FUNCTION - NOT YET IMPLEMENTED

THE SPECIAL FUNCTION STACKADJUST IS A COMPILER DIRECTIVE BY WHICH THE USER ADVISES THE COMPILER HE HAS PERFORMED AN OPERATION WHICH HAS, UNKNOWN TO THE COMPILER, EXPLICITYLY ADJUSTED THE EXECUTION STACK BY A CERTAIN AMOUNT. THE FUNCTION REQUIRES ONE ARGUMENT WHICH EVALUATES TO AN EVEN CONSTANT A COMPILE-TIME. THE VALUE OF THE FUNCTION IS THE VALUE OF THE CONSTANT.

AN EXAMPLE USAGE IS

```
SP←.SP+STACKADJUST(-4)
```

WHICH RESERVES TWO WORD ON THE STACK.

THIS SPECIAL FUNCTION MAY ALSO BE USEFUL IN CONJUNCTION WITH THE INLINE AND EMT/TRAP SPECIAL FUNCTIONS.

II.1.2 MACHINE LANGUAGE

INLINE(TEXTLITERAL)

THE SPECIAL FUNCTION INLINE IS A SIMPLE GENERAL ESCAPE MECHANISM TO ALLOW THE PROGRAMMER TO EMBED MACHINE LANGUAGE CODE IN HIS BLISS-11 PROGRAM. IT'S USE IS HIGHLY MACHINE AND CONTEXT DEPENDANT AND EXTREME CAUTION IS RECOMMENDED. IT IS INTENDED PRIMARILY AS A MEANS TO CONTINUE PROGRAM DEVELOPMENT WHILE A COMPILER BUG OR LANGUAGE DEFICENCY IS BEING HANDLED.

THE TEXT STRING IS SAVED AS IS FOR OUTPUT IN THE COMPILED ASSEMBLY SOURCE. WHEN OUTPUT A CARRIAGE-RETURN/LINE FEED IS APPENDED TO THE LINE. ONLY ONE ASSEMBLY LINE PER INLINE EXPRESSION IS PERMITTED.

A WARNING MESSAGE WILL ALWAYS BE GIVEN WHEN USING THIS CONSTRUCT.

EXAMPLE:

```
%CORE EMT TO DOS %
INLINE ( "MOV #100,-(SP)");
INLINE("EMT 41")
```

II.1.3 COMMUNICATION WITH MONITOR

ADDITIONAL SPECIAL FORMS MAY BE INTRODUCED TO FACILITATE COMMUNICATION WITH THE MONITOR AND/OR IOX.

III. SYSTEM FEATURES

III.1.0 COMPLILATION CONTROL

THE ACTIONS OF THE COMPILER WITH RESPECT TO A PROGRAM MAY BE CONTROLLED BY SPECIFICATIONS A) IN THE INITIAL INPUT STRING FROM A TTY, B) IN THE MODULE HEAD, OR C) BY A SPECIAL SWITCHES DECLARATION. NOT ALL ACTIONS CAN BE CONTROLLED FROM EACH OF THESE PLACES, BUT MANY CAN.
SOME ACTIONS ONCE SPECIFIED HAVE A PERMANENT EFFECT WHILE THE EFFECT OF OTHERS CAN BE MODIFIED (SUCH AS LISTING CONTROL). THE TABLE IN SECTION III.1.4 GIVES A LIST OF VARIOUS COMPILER ACTIONS AND THE ASSOCIATED SWITCH AND/OR SOURCE LANGUAGE CONSTRUCTS WHICH MODIFY THOSE ACTIONS. THIS LIST IS SUBJECT TO CHANGE.

III.1.1 COMMAND SYNTAX

THE GENERAL FORMAT OF THE INITIAL COMMAND TO BLISS-11 IS:

LSTDEV:FILE.EXT←SORCDEV:FILE.EXT,...,SORCDEV:FILE.EXT

THE "LSTDEV:FILE.EXT" MAY BE OMITTED WITH THE IMPLICATION THAT THE CORRESPONDING FILE IS NOT TO BE GENERATED. THE ".EXT" MAY BE OMITTED ON ANY OF THE FILE SPECIFICATIONS AND FOLLOWING DEFAULTS WILL BE ASSUMED:

LISTING FILE:	P11
SOURCE FILE:	B11

AS WITH OTHER DEC CUSP'S, SWITCHES OF THE FORM /X (X=A,B,...,Z) MAY BE PLACED ANYWHERE IN A COMMAND STRING.

THE LISTING FILE IS PROPERLY FORMATTED FOR IMMEDIATE ASSEMBLY BY PALX11 OR MACX11.

III.1.2 MODULE HEAD - NOT YET IMPLEMENTED

AS EXPLAINED IN I.1.1 THE SYNTAX FOR A MODULE IS

MODULE ::= MODULE NAME (PARAMETERS) = E ELUDOM

THE 'PARAMETERS' FIELD MAY CONTAIN VARIOUS INFORMATION WHICH WILL AFFECT THE COMPILER'S ACTION WITH RESPECT TO THE CURRENT PROGRAM. THE SYNTAX OF THIS FIELD IS

PARAMETERS ::= PARAMETER / PARAMETER,PARAMETERS

THE ALLOWED FORMS OF 'PARAMETER' ARE GIVEN IN TABULAR FORM IN SECTION III.1.4 UNDER THE COLUMN HEADED "MODULE HEAD SYNTAX".

WHEN THE COMPILED CODE IS TO BE RUN UNDER THE DOS MONITOR, NO STACK DECLARATION IS REQUIRED. THE DOS LOADER INITIALIZES THE SP REGISTER TO POINT JUST BELOW THE LOWEST WORD OF THE PROGRAM.

III.1.3 SWITCHES DECLARATION

```
DECLARATION ::= SWITCHES SWITCH LIST
SWITCH LIST ::= SWITCH / SWITCH, SWITCH LIST
```

THE SWITCHES DECLARATION ALLOWS THE USER TO SET VARIOUS SWITCHES WHICH CONTROL THE COMPILER'S ACTIONS. THE EFFECT OF A SWITCHES DECLARATION IS LIMITED TO THE SCOPE OF THE BLOCK IN WHICH THE DECLARATION IS MADE. THE VARIOUS ALLOWED FORMS OF 'SWITCH' ARE GIVEN IN TABULAR FORM IN SECTION III.1.4 UNDER THE COLUMN HEADED "SWITCHES DECLARATION".

III.1.4 ACTIONS

COMMAND SWITCH	MODULE HEAD SYNTAX	'SWITCHES' DECLARATION	ACTION
/L	*LIST	*LIST	ENABLE LISTING OF THE SOURCE TEXT. THIS SWITCH IS ASSUMED TRUE INITIALLY.
/-L	NOLIST	NOLIST	DISABLE LISTING OF THE SOURCE TEXT.
/N & /-N	NOERS	NOERS	DO NOT PRINT ERROR MESSAGE ON THE TTY.
/X	SYNTAX	-	SYNTAX CHECK ONLY! NO CODE WILL BE GENERATED - THIS SPEEDS THE COMPILATION PROCESS AND IS THEREFORE USEFUL DURING THE INITIAL STAGES OF PROGRAM DEVELOPMENT.

COMMAND SWITCH	MODULE HEAD SYNTAX	SWITCHES DECLARATION	ACTION
/O	*OPTIMIZE	*OPTIMIZE	BECAUSE OF THE POSSIBILITY OF COMPUTED ADDRESSES IN BLISS PROGRAMS, IT IS NOT POSSIBLE FOR THE COMPILER TO DETERMINE WHETHER OPTIMIZATION OF SUB-EXPRESSIONS IS POSSIBLE ACROSS ";"'S IN A COMPOUND EXPRESSION. THEREFORE THE COMPILER OPERATES IN TWO MODES - ONE IN WHICH IT DOES OPTIMIZE SUCH COMMON SUB EXPRESSIONS AND ONE IN WHICH IT DOES NOT. WHEN THE 'OPTIMIZE' SWITCH IS TRUE THE COMPILER ATTEMPTS TO OPTIMIZE ACROSS A ";". THE DEFAULT MODE IS FOR THE SWITCH TO BE TRUE.
/-O	NOOPTIMIZE	NOOPTIMIZE	SETS THE OPTIMIZATION SWITCH (SEE ABOVE TO FALSE.

*REPRESENTS THE DEFAULT STATE OF MODULE HEAD AND SWITCH DECLARATIONS

III.2 ERROR REPORTING

COMPILER ERROR MESSAGES ARE OF THE FORM:

```
;TYPE#NNN          .........↑.....@.......* L:NNNN1 L:NNNN2 L:NNNN3
;MESSAGE
```

WHERE

TYPE — IS EITHER "WARN" FOR AN INFORMATIONAL OR WARNING MESSAGE AND IS NOT FATAL TO THE COMPILATION, OR

"ERR " FOR A FATAL ERROR. SYNTAX ONLY PROCESSING CONTINUES AFTER A FATAL ERROR MESSAGE.

NNN — IS THE ERROR NUMBER

MESSAGE — IS THE ACTUAL ERROR OR WARNING MESSAGE. (THIS MAY BE SUPPRESSABLE IN THE FUTURE.)

THE SPECIAL CHARACTERS AND LINE NUMBERS FOLLOWING THREE PIECES OF INFORMATION ABOUT THE LOCATION OF THE ERROR:

* AND L:NNNN1 GIVE THE POINT IN THE TEXT WHERE THE ERROR WAS DETECTED,
↑ AND L:NNNN2 GIVE THE BEGINNING OF THE OPENING OF THE CURRENT CONTROL SCOPE IN WHICH THE ERROR IS DETECTED,
@ AND L:NNNN3 GIVE THE END OF THE LAST CONTROL SCOPE SUCCESSFULLY COMPLETED PRIOR TO THIS ERROR.

IF ANY OF THE SPECIAL CHARACTERS OVERLAP, THEN PRIORITY IS GIVEN AS THEY ARE LISTED ABOVE.

EXAMPLE:

```
;00001      BEGIN X←(B←.C;.Y*(.A+.B);.Y Z);

;ERR #066  ........↑.............@....* L:1 L:1 L:1
;MISSING OPERATOR
```

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Carnegie-Mellon University
Department of Computer Science
Pittsburgh, Pennsylvania 15213

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
BLISS/11 REFERENCE MANUAL

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Scientific Interim

5. AUTHOR(S) *(First name, middle initial, last name)*
W. Wulf, J. Apperson, R. Brender, C. Geschke, P. Knueven, C. Weinstock, J. Zarrella, D. Wile

6. REPORT DATE
March 1, 1972

7a. TOTAL NO. OF PAGES
63

7b. NO. OF REFS

8a. CONTRACT OR GRANT NO.
F44620-70-C-0107

b. PROJECT NO.
9769

c. 61102F

d. 681304

9a. ORIGINATOR'S REPORT NUMBER(S)
CMU-CS-72-114

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES
TECH, OTHER

12. SPONSORING MILITARY ACTIVITY
Air Force Office of Scientific Research
1400 Wilson Boulevard
Arlington, Virginia 22209

13. ABSTRACT

This document describes the BLISS implementation language as written for the PDP-11. BLISS is a language specifically designed for use as a tool in implementing large software programs. Special attention is given in the language design to the requirements of the systems programming task, such as: space and time efficiency, the representation of data structures, the lack of run-time support facilities, flexible control structures, modularization, and parameterization of programs.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT