Memorandum

SINGER
BUSINESS MACHINES

to: TREB Board

SAN LEANDRO, CALIFORNIA

from: B. Cunningham

refer to:

subject: Approved ERS for Single K01

date: 12/27/74

The External Reference Specifications for Single Cross-Compiler have been approved and are attached, along with the recommendations of the TREB chairman.
Additional copies of the attached documentation are available from Internal Document Distribution.

Bev Cunningham
IDD

cc:

C. Bradley	TREB
K. Dunham	
E. Hay	
W. Powell	
P. Thompson	
D. Quitslund	
R. Carmichael	w/o
G. Broyles	w/o
L. Fesperman	
R. Gunther	w/o
W. Holloway	
L. Krider	
H. Lam	(3)
J. Landau	
D. Lemos	w/o
C. Miller	
R. Proctor	w/o
R. Ogg	w/o
R. Zemlin	

FORM 6006

Memorandum

SINGER
BUSINESS MACHINES

to: C. E. Miller

SAN LEANDRO, CALIFORNIA

from: W. R. Powell

refer to:

subject: ERS for SINGLE (K01)

date: 12/18/74

The TREB for Project K01 has completed the review of the ERS for SINGLE. Unlike most reviews by a TREB, our efforts have spanned several months. During that time, several changes, enhancements, revisions, etc., were requested and I believe we finally have a product which will be satisfactory for the initial implementation.

There remains, however, a few items that should be detailed here. These items should be considered as either requiring additional details (possibly in appendix form) or are items you should be aware of in dealing with this product in the future:

- The details of all error messages (format, content, etc.) have not been specified in the ERS, but we expect them to be included in a subsequent appendix.
- The print character set is satisfactory for our implementation, but may not be acceptable for a European implementation.
- The ability to alter the syntax of the "analyzer" is referred to in the document, but is not clearly defined in the ERS.

The above items do not alter the basis of our recommendation, but only serve as points of information.

Several enhancements that were **requested** by the TREB were declared not feasible for the initial implementation. This was due not only to the time constraints for the initial implementation, but also in part to design philosophy differences between the TREB and the developers. I have, however, taken the liberty of including a list of enhancements in this memo that the TREB feels must be implemented in order to make the product totally usable by Programming Systems. This list is arranged in priority sequence as established by the TREB.

1. Allow constant and preset records.
2. Provide clearer definition of asynchronous processing capabilities.
3. Implement overlay capability.
4. String, SINT and INT qualifiers default to value size for constant and preset items.

5. Add ability to generate re-entrant code.

6. Allow maximum string length of 2048 characters.

7. Allow definition of names to substitute for qualified reference forms.

8. Allow literals in simple type qualifiers within the constant section.

9. Maximum size for numeric data and maximum length of numeric literals = 18 decimal digits. Also allow arithmetic on operands up to 18 digits long.

10. Add a source construct expressing an 'edit' function.

11. Allow numeric literals in all type qualifiers.

12. Add pointer variables.

13. Allow (or make mandatory) explicit specification array lower subscript limits.

14. Allow literal values in action statements.

15. Implement compiler on System Ten.

16. Add machine-independent console I/O capability.

17. Allow conditional expressions.

18. Add binary and packed decimal data items.

19. Consider the ability to use coroutines.

20. Add hexidecimal data items.

The priority listed above is a "recommended priority" and reflects the opinion of the TREB. We recognize the final selection is at the discretion of management.

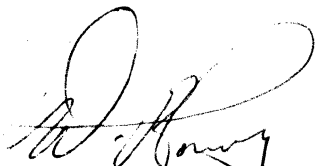
Some of these recommendations concern the "ease of use" of the language. Only the use of the language will determine if the "ease of use" factor is outweighed in practice by the extra structure.

Other recommendations concern the ability to use the language for all programs or products. The TREB recommends that SINGLE be used on products only after due consideration is made of the language limitations with respect to the program or product to be developed.

Since the above list functions as an "RSM list", I recommend we create a formal RSM for each enhancement and process the RSMs per our existing RSM procedure. In order to adequately describe the details of the RSMs, consideration should be made to include a copy of each RSM with any distribution of the ERS. I also recommend this memo be included in all distributions of the ERS. This would assure the readers of the importance placed upon the RSM list by the TREB.

One of the duties of the TREB is to point out discrepancies between a product's POR and ERS. These discrepancies may be trivial, but they should, nevertheless, be called to your attention. The POR states the compiler runs on the IBM 370 OS/MVT. The ERS reflects an update to this by stating the compiler runs on the IBM 370 OS/VS2. I believe this update is due to the current operating environment at SISCO. Per our previous discussions in which this point was raised and your willingness to accept this discrepancy, I recommend you approve the ERS for the initial implementation.

It is the understanding of the TREB, that unlike the usual automatic termination of the TREB duties upon issuing a letter of recommendation, this TREB will remain "intact" to provide recommendations to the enhancements detailed above.



W. R. Powell
TREB Chairman

WRP:cmc
cc: L. Fesperman
D. Quitslund
E. Hay
K. Dunham
C. Bradley
P. Thompson
R. Zemlin
L. Krider
R. Gunther
J. Landau
W. Holloway
H. Lam

REVISIONS			SHEET REVISION STATUS									
REV	CHANGE DOCUMENT	BY	SH	REV	SH	REV	SH	REV	SH	REV	SH	REV
			1	A	25	A	49	A	4c	A	28c	A
			2	A	26	A	50	A	5c	A	29c	A
			3	A	27	A	51	A	6c	A	30c	A
			4	A	28	A	52	A	7c	A	31c	A
			5	A	29	A	1a	A	8c	A	32c	A
			6	A	30	A	2a	A	9c	A	33c	A
			7	A	31	A	3a	A	10c	A	34c	A
			8	A	32	A	4a	A	11c	A	35c	A
			9	A	33	A	5a	A	12c	A	36c	A
			10	A	34	A	6a	A	13c	A	37c	A
			11	A	35	A	7a	A	14c	A	38c	A
			12	A	36	A	1b	A	15c	A	39c	A
			13	A	37	A	2b	A	16c	A		
			14	A	38	A	3b	A	17c	A		
			15	A	39	A	4b	A	18c	A		
			16	A	40	A	5b	A	19c	A		
			17	A	41	A	6b	A	20c	A		
			18	A	42	A	7b	A	21c	A		
			19	A	43	A	8b	A	22c	A		
			20	A	44	A	9b	A	23c	A		
			21	A	45	A	10b	A	24c	A		
			22	A	46	A	11b	A	25c	A		
			23	A	47	A	1c	A	26c	A		
			24	A	48	A	2c	A	27c	A		
							3c	A				

APPROVED

EXTERNAL REFERENCE SPECIFICATION

SINGLE

ORIGINAL SPECIFICATION - REVISION A

FIRST USED ON	
MODEL	ASSEMBLY

DRAWN BY	RAZ	DATE 12/20/74
CHECKED		12/20/74
ENGRG		12/20/74
MFG		

SINGER
BUSINESS MACHINES

ALBUQUERQUE, NEW MEXICO 87103
NIJMEGEN, HOLLAND
SAN LEANDRO, CALIFORNIA 94577

SIZE	CODE IDENT NO.	
A	23610	K01

COVER SHEET

TOTAL SHEETS	NEXT SHEET	SHEET NO.
92	2	1

1. PRODUCT DESCRIPTION

1.1 PRODUCT NAMES AND NUMBERS

1.1.1 PRODUCT NAME

SINGLE CROSS-COMPILER

1.1.2 NAME ABBREVIATIONS

SINGLE

1.1.3 PROJECT NUMBERS

K01

1.1.4 PRODUCT ID NUMBERS

G005

1.2 BRIEF DESCRIPTION OF PRODUCT

THIS PRODUCT IS A COMPILER WHICH RUNS ON AN IBM 370 OS/VS2 SYSTEM. IT ACCEPTS AS SOURCE LANGUAGE THE SOFTWARE IMPLEMENTATION LANGUAGE CALLED SINGLE, AND PRODUCES OBJECT CODE FOR THE SINGER SYSTEM TEN COMPUTER, IN A FORM ACCEPTABLE TO THE SYSTEM TEN LINK EDITOR PROGRAM.
CERTAIN FEATURES OF THE SINGLE LANGUAGE ARE DEFERRED IN THIS IMPLEMENTATION. THESE ARE:

- OVERLAY HANDLING
- ASYNCHRONOUS PROCESSING
- MPU I/O INSTRUCTIONS

SUBSEQUENT VERSIONS OF THIS COMPILER ARE PLANNED TO RUN ON THE SYSTEM TEN, TO GENERATE MPU OBJECT CODE, ETC. EACH OF THESE WILL BE SPECIFIED IN A SEPARATE ERS.

1.2.1 TECHNICAL REFERENCES

A DESCRIPTION OF SINGLE, THE SINGER IMPLEMENTATION LANGUAGE NOVEMBER, 1974. (COPY ATTACHED).

A COMPILER GENERATOR, BY MCKEEMAN ET. AL., PRENTICE-HALL, 1970.

SYSTEM TEN LINK EDITOR REFERENCE MANUAL, 44-8476-401-00.

SYSTEM TEN RJF DISC SYSTEM REFERENCE MANUAL, 40-315.

DMF II PROGRAMMER'S REFERENCE MANUAL, 44-B800-401-00.

OS JCL JOB CONTROL LANGUAGE REFERENCE GC28-6704-3

HASP II SYSTEM MANUAL 360D-05.1.014 VERSION 3

PL/I LANGUAGE REFERENCE MANUAL, GC33-0009.

CPU INSTRUCTIONS REFERENCE MANUAL 42-2500-01.

MODEL 21 PROCESSOR REFERENCE MANUAL, 42-2022.

MODEL 21 ACU, 9001500-01, FUNCTIONAL AND PHYSICAL SPECIFICATION 9007033

MODEL 20 ACU FUNCTIONAL SPECIFICATION, 9006681.

2. POR RECONCILIATION

ALL CONSTRAINTS ON THE EXTERNAL DESIGN OF THIS PRODUCT ARE DEFINED BY THIS ERS AND THE POR'S/ERS'S LISTED BELOW. THE POR'S TAKE PRECEDENCE OVER THIS ERS IN EVERY INSTANCE. APPARENT CONFLICTS OR AMBIGUITIES SHALL BE INVESTIGATED AS SOON AS THEY ARE DETECTED. CONFIRMED CONFLICTS BETWEEN AND ANY OVERSIGHTS IN ANY OF THE DOCUMENTS SHALL BE BROUGHT TO THE ATTENTION OF THE DIRECTOR, PROGRAMMING SYSTEMS IMMEDIATELY.

THIS ERS SHALL DETAIL ALL FEATURES STATED IN THE POR'S AND RELATED ERS'S. AS SUCH, IT IS THE FINAL EXTERNAL LEVEL DOCUMENT FOR DEVELOPMENT PURPOSE AND IN NO WAY EXTENDS THE SCOPE OF THE PRODUCT.

OMISSION OF FEATURES INCLUDED IN THE POR'S OR ADDITION OF FEATURES IS NOT PERMITTED.

2.1 OTHER RELEVANT POR(S) AND ERS(S)

THE POR CALLS FOR A NUMBER OF PRODUCTS, GENERATING CODE FOR SEVERAL OBJECT MACHINES AND INCORPORATING CERTAIN FEATURES OF THE SINGLE LANGUAGE WHICH ARE OMITTED FROM THIS FIRST IMPLEMENTATION. THESE PRODUCTS WILL BE DETAILED IN FUTURE ERS DOCUMENTS, EACH OF WHICH IS CONSTRAINED TO BE UPWARD COMPATIBLE WITH THIS ERS, IN THE SENSE THAT THAT COMPILER WILL ACCEPT ANY PROGRAM WHICH CONFORMS TO THE SPECIFICATIONS OF THIS ERS.

3. CONVENTIONS

SECTION 5.2.3.3 CONSTITUTES A FORMAL DEFINITION OF THE SINGLE LANGUAGE. FOR A MORE INFORMAL EXPOSURE TO THE BASIC IDEAS AND STRUCTURE OF THE LANGUAGE, THE READER UNACQUAINTED WITH SINGLE IS ADVISED TO FIRST READ THE ATTACHED REFERENCE DOCUMENT, 'A DESCRIPTION OF SINGLE.'

3.1 NOTATION

THE BASIC NOTATION FOR THE SYNTAX RULES OF SECTION 5.2.3.3 IS THE STANDARD METALANGUAGE FOR DESCRIBING SYNTAX OF CONTEXT-FREE GRAMMARS. IT MAKES USE OF FOUR METASYMBOLS:

< AND > ARE USED TO BOUND THE NAME OF A SYNTACTIC CONSTRUCT, FOR EXAMPLE, THE APPEARANCE OF <SIMPLE TYPE> IN A SYNTAX RULE MEANS: ANY SEQUENCE OF INPUT CHARACTERS WHICH HAS BEEN FOUND TO SATISFY THE DEFINITION OF THE CONSTRUCT "SIMPLE TYPE".

::= IS THE METASYMBOL USED TO DENOTE THE DEFINTION OF THE CONSTRUCT WHOSE NAME APPEARS TO THE LEFT OF THIS SYMBOL AS AN INSTANCE OF THE FORM APPEARING TO THE RIGHT OF THE SYMBOL. FOR EXAMPLE,

```
<SIMPLE TYPE> ::= <IDENTIFIER>
```

! IS THE METASYMBOL DENOTING ALTERNATE FORMS WHICH MAY BE USED TO DEFINE A CONSTRUCT. FOR EXAMPLE,

```
<SIMPLE TYPE> ::= <IDENTIFIER>
                ! INT<PQUALIF>
```

SPECIFIES TWO ALTERNATIVE WAYS OF FORMING AN INSTANCE OF <SIMPLE TYPE>. NOTE THAT SYNTAX DEFINITIONS NEED NOT BE EXCLUSIVE, AND THEREFORE THE ABOVE EXAMPLE IS SIMPLY SHORTHAND FOR THE PAIR OF DEFINITIONS

```
<SIMPLE TYPE> ::= <IDENTIFIER>
<SIMPLE TYPE> ::= INT <PQUALIF>
```

3.2 DEFINITIONS

3.2.1 SYNTAX RULES:

EACH RULE OF THE SYNTAX IS EXPRESSED AS

<CONSTRUCT NAME> ::= FORM

OR, FOLLOWING ANOTHER RULE FOR THE SAME CONSTRUCT, AS

! FORM

A FORM IS A SEQUENCE OF LITERAL CHARACTER STRINGS AND/OR CONSTRUCT NAMES BOUNDED BY THE METASYMBOLS < AND >. ANY SEQUENCE OF CHARACTERS PRECEEDED BY < AND SUCCEEDED BY > IS A CONSTRUCT NAME. ANY SEQUENCE OF NON-BLANK CHARACTERS NOT BOUNDED BY THESE METASYMBOLS IS A LITERAL STRING, REPRESENTING THE LITERAL APPEARANCE OF THOSE CHARACTERS IN THE INPUT TEXT.

BLANK CHARACTERS BEGINNING OR TERMINATING A FORM ARE NOT SIGNIFICANT, NOR ARE BLANK CHARACTERS BETWEEN TWO CONSTRUCTS OR BETWEEN A LITERAL STRING AND A CONSTRUCT. BLANK CHARACTERS BETWEEN TWO SUCCESSIVE LITERAL STRINGS IMPLY THAT SOME PURE SEPARATOR MUST APPEAR IN THE INPUT TEXT BETWEEN THE TWO STRINGS. (SEE BELOW FOR DISCUSSION OF SEPARATORS)

THE MEANING OF EACH SYNTAX RULE IS THAT IF A SEQUENCE OF CHARACTERS IN THE INPUT TEXT IS FOUND (IGNORING NON-SYNTACTIC SEPARATORS) TO CONSIST OF STRINGS WHICH "SATISFY" THE FORM ON THE RIGHT SIDE OF THE ::=, THEN THAT SEQUENCE OF TEXT MAY BE CONSIDERED AN INSTANCE OF THE CONSTRUCT WHOSE NAME APPEARS ON THE LEFT OF THE ::=. A FORM IS "SATISFIED" IF EACH SUCCESSIVE INPUT STRING SATISFIES THE SUCCESSIVE ELEMENTS OF THE FORM, IN THE SENSE THAT:

- A LITERAL IN A FORM IS SATISFIED BY THE EXACT APPEARANCE OF THAT SEQUENCE OF CHARACTERS IN THE INPUT TEXT;
- A CONSTRUCT IN A FORM IS SATISFIED BY THE APPEARANCE OF A SEQUENCE OF INPUT TEXT WHICH SATISFIES ONE OF THE FORMS WHICH DEFINE THAT CONSTRUCT.

FOR EXAMPLE, THE RULE FOR REPLACEMENT STATEMENTS:

<REP STAT> ::= <STORE LIST><REPLACE><EXPRES>

IS SATISFIED BY THE INPUT TEXT

A1,A2:=X*(Y+Z)

WHERE THE STRING A1,A2 SATISFIES <STORE LIST>, THE STRING := SATISFIES <REPLACE>, AND THE REMAINDER SATISFIES <EXPRES>.

3.2.2 LOW LEVEL SYNTAX - THE SCANNER

WITHIN SECTION 3.2.2, REFERENCE TO A SPECIFIC CHARACTER OR SEQUENCE OF CHARACTERS WILL FOLLOW THE RULES OF THE SINGLE LANGUAGE FOR THE SPECIFICATION OF SUCH LITERAL STRINGS: THE REPRESENTATION OF A LITERAL STRING IS DENOTED BY THE APPEARANCE OF ONE OF THE CHARACTERS APOSTROPHE OR QUOTE MARK, AND ALL CHARACTERS FOLLOWING UNTIL THE NEXT APPEARANCE OF THAT SAME DELIMITER FORM A LITERAL STRING.

BELOW THE LEVEL OF SYNTAX IN A CONTEXT-FREE GRAMMAR IS THE SCANNER LEVEL WHICH RECOGNIZES THE TERMINAL CONSTRUCTS, I.E., THOSE CONSTRUCTS IN THE GRAMMAR WHICH HAVE NO DEFINING RULE. FOR EXAMPLE, THE CONSTRUCTS <IDENTIFIER> AND <NUMBER> ARE TERMINAL CONSTRUCTS. THEY APPEAR IN FORMS DEFINING OTHER CONSTRUCTS BUT ARE NOT THEMSELVES DEFINED WITHIN THE RULES. THE REQUIREMENTS AND RESTRICTIONS IMPOSED BY THE SCANNER ARE DISCUSSED IN PART IN 5.2.3.3.2, SEMANTICS NOTES, WHICH IS ORGANIZED TO PARALLEL THE SYNTAX RULES OF 5.2.3.3.1. HOWEVER, SOME ASPECTS OF THE SCANNER ENCOMPASS ALL OF THE SYNTAX RULES, AND ARE MORE CONVENIENTLY DISCUSSED HERE.

3.2.2.1 SEPARATORS

IN RECOGNIZING THE TERMINAL CONSTRUCTS OF A LANGUAGE, SOME MEANS IS NEEDED TO DISTINGUISH THE BEGINNING AND ENDING OF EACH CONSTRUCT WITHOUT LOOKING AHEAD OR BACK IN THE INPUT TEXT. FOR EXAMPLE, IF THE INPUT TEXT INCLUDES:

IF A = B THEN.....

THE SCANNER MUST RECOGNIZE THAT "IF" IS A TERMINAL CONSTRUCT

(RESERVED WORD) AND "A" IS AN INSTANCE OF <IDENTIFIER>. THIS LEADS TO THE IDEA OF SEPARATORS, WHICH NEED NOT BE PART OF THE SYNTAX BUT SERVE TO NOTIFY THE SCANNER THAT ANY TERMINAL CONSTRUCT BEGUN TO THE LEFT OF THE SEPARATOR ENDS BEFORE THE SEPARATOR. THE SCANNER FOR THE SINGLE LANGUAGE RECOGNIZES THREE PURE (THAT IS, SERVING NO OTHER FUNCTION) SEPARATORS:

" "(BLANK),
";", AND
END-OF-LINE.

THE USE OF ";" AS A SEPARATOR IS MEANT TO EASE THE TRANSITION FOR PROGRAMMERS WITH ALGOL OR PL/I EXPERIENCE. ALTHOUGH ";" IS ACCEPTED AS A SEPARATOR, IT WILL NOT APPEAR IN SOURCE LANGUAGE OUTPUT, BEING REPLACED AS APPROPRIATE BY " " OR END-OF-LINE.

THE USE OF END-OF-LINE AS A SEPARATOR DOES NOT IMPLY THAT STATEMENTS OR EXPRESSIONS MAY NOT BE CONTINUED FROM ONE LINE TO THE NEXT, BUT ONLY THAT TERMINAL CONSTRUCTS SUCH AS <IDENTIFIER>, RESERVED WORDS, ETC., MUST BEGIN AND END ON THE SAME LINE. SEE 3.2.2.3 FOR SPECIFIC SEPARATOR REQUIREMENTS.

TWO OTHER CHARACTERS, "$" AND "?", FUNCTION ALSO AS SEPARATORS BUT HAVE ADDITIONAL NON-SYNTACTIC FUNCTIONS:

"$" INDICATES THAT THE '$' AND ALL SUBSEQUENT CHARACTERS UNTIL NEXT END-OF-LINE FORM A COMMENT, NOT PART OF THE INPUT SYNTAX, BUT TO BE PRINTED IN THE SOURCE OUTPUT LISTING. ANY CHARACTERS OF THE SINGLE LANGUAGE MAY APPEAR WITHIN A COMMENT. TRAILING BLANKS, AND IN SOME CASES LEADING BLANKS, MAY BE SUPPRESSED IN THE OUTPUT OF A COMMENT. NO INTERPRETATION IS MADE OF NON-BLANK CHARACTERS WITHIN A COMMENT.

"?" IS USED AS A DEBUGGING CONTROL FOR THE COMPILER ITSELF. IT ACTS LIKE A COMMENT IN THAT EVERYTHING FROM "?" TO END-OF-LINE IS NON-SYNTACTIC, AND GETS PRINTED IN THE SAME MANNER AS A COMMENT. HOWEVER, THE CHARACTERS FOLLOWING THE "?" ARE ANALYZED, AND VARIOUS COMPILER DEBUG OPTIONS ARE CONTROLLED BY THEM. THIS IS NOT AN EXTERNAL FEATURE, AND THE INTERPRETATION OF THE CHARACTERS IN A DEBUG CONTROL ELEMENT IS SUBJECT TO CHANGE HOWEVER AND WHENEVER THE COMPILER MAINTENANCE FUNCTION DEEMS IT NECESSARY. THUS THE USER SHOULD REGARD THE EFFECT OF '?' AS UNDEFINED.

3.2.2.2 CHARACTER SET

THE SINGLE CHARACTER SET CONSISTS OF ALL BUT TWO OF THE ASCII 64 CHARACTER SET. THE EXCLUDED CHARACTERS ARE "%" AND "&". THESE CHARACTERS ARE REQUIRED FOR COMMUNICATIONS PURPOSES, AND THEIR APPEARANCE WITHIN SINGLE INPUT TEXT WILL RESULT IN THEIR REPLACEMENT BY "↑" AND "\" RESPECTIVELY.

THE SINGLE CHARACTERS CONSIST OF 27 <ALPHABETIC> CHARACTERS:

"ABCDEFGHIJKLMNOPQRSTUVWXYZ←"

PLUS 10 DIGITS:

"0123456789"

PLUS 2 SEPARATORS:

" ;"

PLUS 5 OPERATORS:

"+-*/\"

PLUS 4 RELATIONAL SYMBOLS:

"<>=↑"

PLUS 6 GROUPING SYMBOLS:

"()[]'" AND '"'

PLUS 6 MISCELLANEOUS SYMBOLS:

"£.,:!@"

PLUS 2 SPECIAL SEPARATORS:

"$?"

THE APPEARANCE OF ANY CHARACTER NOT ALPHABETIC OR A DIGIT SERVES THE FUNCTION OF A SEPARATOR IN ADDITION TO WHATEVER SYNTACTIC AND/OR SEMANTIC SIGNIFICANCE IT MAY HAVE. NOTE THAT ALTHOUGH THE SCANNER NEVER NEEDS MORE THAN A SINGLE SEPARATOR BETWEEN TWO TERMINAL CONSTRUCTS, IT WILL ACCEPT ANY NUMBER OF NON-SYNTACTIC SEPARATORS IN SUCCESSION. THUS, FOR EXAMPLE, IT IS PERMISSIBLE, ALTHOUGH REPREHENSIBLE, TO

WRITE:

```
X:=A;    $ .......
$.................
$..........
$..............  (LOTS AND LOTS OF COMMENTS)
$..............
  ; + ; $......  (MORE COMMENT)
  ; C ; $......
```

THE SCANNER WILL PASS

```
X:=A+C
```

ON TO THE SYNTAX ANALYZER.

ALL OF THE NON-ALPHABETIC AND NON-DIGIT CHARACTERS ARE EITHER NON-SYNTACTIC, " ;$?", OR ARE THEMSELVES TERMINAL CONSTRUCTS. THE RECOGNITION OF MULTIPLE SYMBOL CONSTRUCTS SUCH AS ":=", "=>",ETC. IS NOT MADE BY THE SCANNER, AND IS NOT AFFECTED BY NON-SYNTACTIC SEPARATORS, ALTHOUGH ON OUTPUT SUCH CONSTRUCTS WILL HAVE NO PURE SEPARATORS BETWEEN THEIR COMPONENT CHARACTERS.

3.2.2.3 COMPOSITE TERMINAL CONSTRUCTS AND PURE SEPARATORS

SEE APPENDIX C, TERMINAL AND NON-TERMINAL CONSTRUCTS, FOR A LISTING OF THE TERMINAL CONSTRUCTS OF THE SINGLE LANGUAGE. THEY CONSIST OF "<!=", INDIVIDUAL SPECIAL CHARACTERS, PURELY ALPHABETIC LITERALS (RESERVED WORDS), AND THE FOUR CONSTRUCTS

<IDENTIFIER>
<NUMBER>
<QSTRING>
<EMPTY>

OF THESE, <EMPTY> IS A CONDITION RECOGNIZED BY THE HIGHER LEVEL SYNTAX AND IS OF NO CONCERN TO THE SCANNER.

THE REQUIREMENTS FOR PURE SEPARATORS BETWEEN TERMINAL CONSTRUCTS MAY BE STATED AS FOLLOWS:

A PURE SEPARATOR IS REQUIRED BETWEEN TWO TERMINAL CONSTRUCTS IF AND ONLY IF THE FIRST IS AN IDENTIFIER OR A

RESERVED WORD, AND THE SECOND IS AN IDENTIFIER, A RESERVED WORD, OR A NUMBER.

3.2.2.4 RULES FOR COMPOSITE TERMINAL CONSTRUCTS.

A COMPOSITE TERMINAL CONSTRUCT HAS AS ITS LEFTMOST CHARACTER EITHER AN ALPHABETIC, A DIGIT, OR ONE OF THE STRING QUOTES '"' OR "'". IN THE FIRST CASE, THE CONSTRUCT IS EITHER A RESERVED WORD OR AN IDENTIFIER. THE SCANNER EXAMINES SUCCESSIVE CHARACTERS AND TERMINATES THE CONSTRUCT WHEN END-OF-LINE IS ENCOUNTERED OR WHEN THE NEXT CHARACTER IN THE LINE IS NEITHER AN ALPHABETIC NOR A DIGIT. IF THE RESULT HAS MORE THAN 10 CHARACTERS, AN ERROR MESSAGE IS GENERATED, AND ALL BUT THE LEFT-MOST 10 CHARACTERS ARE DISCARDED. THE CONSTRUCT IS COMPARED WITH THE LIST OF RESERVED WORDS, AND IS IDENTIFIED AS THAT RESERVED WORD IF A MATCH IS FOUND. IF NO MATCH IS FOUND, THE CONSTRUCT IS IDENTIFIED AS <IDENTIFIER>. IN THIS IMPLEMENTATION, IDENTIFIERS MAY BE NO MORE THAN SIX CHARACTERS IN LENGTH (FIVE CHARACTERS IF PRECEDED BY '£'). IF THERE ARE FEWER THAN 6 (OR 5) CHARACTERS TO BEGIN, BLANKS ARE ADDED ON THE RIGHT TO FILL. NOTE THAT THE FULL SINGLE LANGUAGE ALLOWS 10-CHARACTER IDENTIFIERS. THE RESTRICTION TO 6 IS FOR CONSISTENCY WITH OTHER SYSTEM TEN CONVENTIONS. IF THE IDENTIFIER WAS LONGER THAN SIX CHARACTERS (OR FIVE IF PRECEDED BY A '£'), AND NO ERRORS HAVE BEEN ENCOUNTERED, A WARNING MESSAGE RESULTS, AND THE OUTPUT LISTING WILL HAVE THE IDENTIFIER TRUNCATED ON THE RIGHT TO THE MAXIMUM LENGTH. IN ANY EVENT THE COMPILER WILL PLACE THE TRUNCATED VERSION OF THE IDENTIFIER IN THE SYMBOL TABLE.

IF THE LEFTMOST CHARACTER IS A DIGIT, THE CONSTRUCT IS IDENTIFIED AS <NUMBER>, AND THE SCANNER EXAMINES SUCCESSIVE CHARACTERS AND TERMINATES THE CONSTRUCT WHEN THE NEXT CHARACTER IS NOT A DIGIT, OR WHEN END-OF-LINE IS FOUND. IF THE RESULT HAS MORE THAN 10 DIGITS, AN ERROR MESSAGE IS GENERATED, AND ALL BUT THE RIGHTMOST 10 DIGITS ARE DISCARDED.

IF THE LEFTMOST CHARACTER IS A STRING QUOTE, THE CONSTRUCT IS IDENTIFIED AS <QSTRING>. THE SCANNER EXAMINES SUCCESSIVE CHARACTERS UNTIL THE SAME QUOTE CHARACTER IS FOUND, OR UNTIL END-OF-LINE. IN THE LATTER CASE AN ERROR MESSAGE IS GENERATED. IF THE CONSTRUCT HAS MORE THAN 60 CHARACTERS (NOT COUNTING THE DELIMITERS) AN ERROR MESSAGE IS GENERATED, AND ALL BUT THE LEFTMOST 60 CHARACTERS ARE DISCARDED.

3.2.2.5 POSITION-DEPENDENT ELEMENTS.

CERTAIN ASPECTS OF LOW-LEVEL SYNTAX DEPEND UPON POSITION IN THE INPUT LINE. THESE ARE:

A) THE BLOCK TERMINATORS,'←!←' AND '/*' (SEE 5.2.3.3.2, RULES 1:2) ARE RECOGNIZED ONLY IF THEY APPEAR STARTING IN COLUMN 1 OF AN INPUT LINE. IF THEY APPEAR ELSEWHERE WITHIN A LINE, ONE OR MORE ERROR MESSAGES MAY RESULT, AND THE SCANNER WILL ATTEMPT TO READ FURTHER.

B) ONLY THE FIRST 74 CHARACTERS OF EACH INPUT LINE ARE READ. ANY ADDITIONAL CHARACTERS IN THE LINE ARE IGNORED. THE SOURCE OUTPUT LISTING CONSISTS OF 78-CHARACTER RECORDS OF WHICH THE LAST FOUR CHARACTERS ARE A FOUR DIGIT LINE NUMBER CALCULATED BY THE COMPILER, FOR USE AS A BASIS FOR CROSS-REFERENCE TABLES.

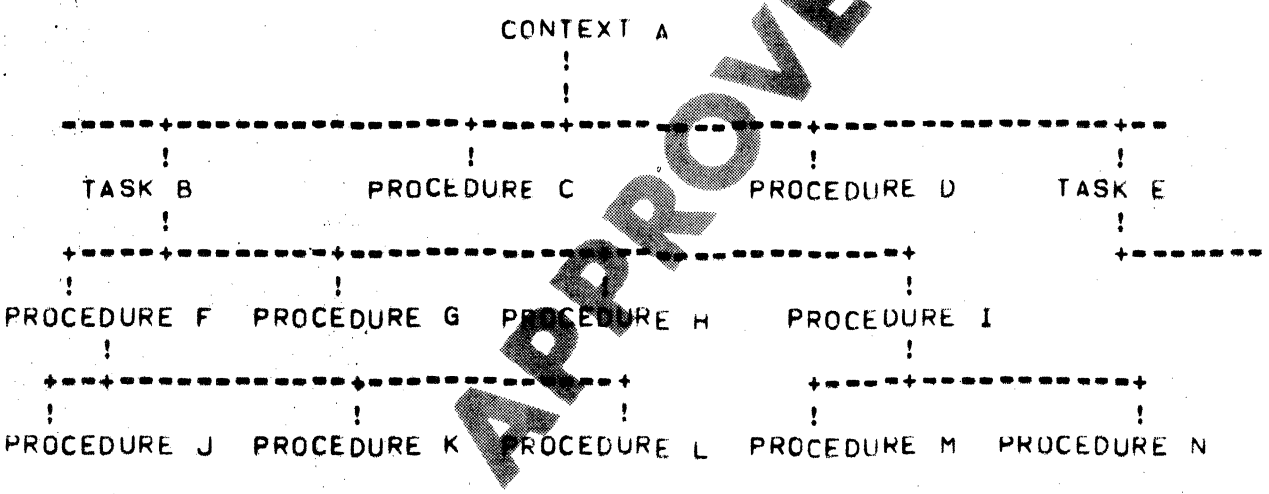
C) IF THE FIRST CHARACTER OF AN INPUT LINE IS A DIGIT, THEN ALL SUCCEEDING CHARACTERS UP TO THE FIRST BLANK WILL BE IGNORED BY THE SCANNER. THIS IS DONE SO THAT THE COMPILER CAN ACCEPT AN INPUT TEXT WHICH CONTAINS LEVEL AND CONTROL INFORMATION SUCH AS THAT PRODUCED BY THE COMPILER SOURCE LISTING. IF NO BLANK IS FOUND IN SUCH A LINE, AN ERROR MESSAGE WILL BE GENERATED.
ALSO, IF THE FIRST CHARACTER OF THE LINE IS A DIGIT, AND THE NEXT IS AN ASTERISK, THE ENTIRE LINE WILL BE IGNORED WITH NO MESSAGE GENERATED. THIS CONVENTION ALLOWS MANY NON-ERROR MESSAGES TO BE INPUT TO THE COMPILER WITH NO ADVERSE EFFECTS.

D) VARIOUS UNDOCUMENTED CONVENTIONS ARE USED IN THE RJF DISC-HASP COMMUNICATION, INVOLVING SPECIAL CHARACTERS IN COL. 1 OF AN INPUT OR OUTPUT LINE. IF THE RJF IS TO BE USED, A SAFE POLICY IS TO INSURE THAT COL. 1 OF EACH INPUT LINE CONTAINS AN ALPHABETIC (INCLUDING '←'), A NUMERIC, '*!, OR A BLANK. THE COMPILER OUTPUT IS STRUCTURED TO AVOID ANY PROBLEMS OF THIS TYPE.

3.2.3 STRUCTURAL TERMS

THE STRUCTURE OF A SINGLE PROGRAM MAKES USE OF A NUMBER OF CONCEPTS FOR WHICH THE TERMINOLOGY IS DESCRIBED HERE.
AN INDIVIDUALLY COMPILABLE UNIT OF SOURCE CODE IS TERMED A BLOCK. A BLOCK IS EITHER A PROGRAM BLOCK OR A CONTEXT BLOCK.
A PROGRAM BLOCK IS EITHER A PROCEDURE BLOCK OR A TASK BLOCK.

A COMPLETE PROGRAM CONSISTS OF A SET OF BLOCKS: A CONTEXT BLOCK, A TASK BLOCK, AND POSSIBLY OTHER PROGRAM BLOCKS. (IN THE CURRENT IMPLEMENTATION, THESE OTHER PROGRAM BLOCKS MUST BE PROCEDURE BLOCKS). A CONTEXT BLOCK MAY BE COMMON TO SEVERAL DISTINCT COMPLETE PROGRAMS. EACH TASK BLOCK TO WHICH THE CONTEXT BLOCK DEFINES LINKS IS THE HIGHEST LEVEL OF SOME COMPLETE PROGRAM.
THROUGH DEFINITIONS, EACH BLOCK IS LINKED TO A UNIQUE HIGHER LEVEL BLOCK, EXCEPT THAT A CONTEXT BLOCK HAS NO HIGHER LEVEL. ALSO, EACH BLOCK DEFINES LINKS TO LOWER LEVEL BLOCKS. EACH OF THESE LOWER LEVEL BLOCKS MUST LINK UPWARD TO THE BLOCK WHICH DEFINED DOWNWARD LINKS TO THEM. THUS A TYPICAL PROGRAM STRUCTURE MIGHT HAVE:



HERE TASK B IS THE HIGHEST LEVEL BLOCK OF A COMPLETE PROGRAM. TASK E IS THE HIGHEST LEVEL BLOCK OF ANOTHER COMPLETE PROGRAM WHICH MAKES USE OF THE SAME CONTEXT. PROCEDURES C AND D ARE ACCESSIBLE TO ANY PROGRAM USING CONTEXT A, AND WILL BE LOADED AS PART OF ANY SUCH PROGRAM. A BLOCK IS CALLED THE PARENT OF ANY BLOCK LINKED TO IT AT THE NEXT LOWER LEVEL. THUS, FOR EXAMPLE, F IS THE PARENT OF J, K, AND L. J, K, AND L ARE SAID TO BE DIRECTLY SUBORDINATE TO F. IF TWO BLOCKS, SUCH AS B AND K, ARE SUCH THAT THERE EXISTS A SEQUENCE OF BLOCKS B, X, Y, . . K WITH EACH BLOCK THE PARENT OF THE FOLLOWING BLOCK, THEN K IS SAID TO BE SUBORDINATE TO B. FOR ANY BLOCK, THE SCOPE OF THE BLOCK IS DEFINED AS THE BLOCK ITSELF PLUS ALL BLOCKS WHICH ARE SUBORDINATE TO IT.

4. PRODUCT ORGANIZATION

THE MODULE STRUCTURE OF THIS PRODUCT IS NOT AN EXTERNAL FEATURE, AND IS NOT DISCUSSED IN THIS ERS. THE DISCUSSION IN SECTION 5 DESCRIBES ONE UNIFIED USER INTERFACE TO THE COMPILER AND ITS ENVIRONMENT.

THE SINGLE CROSS-COMPILER OPERATES ON ANY IBM 370 CAPABLE OF RUNNING OS/VS2 AND PL/I LEVEL H. IT PRODUCES A SOURCE OUTPUT LISTING, VARIOUS OPTIONAL REFERENCE LISTINGS, AND OPTIONAL COMPILED BLOCKS OF TWO KINDS: A LOAD MODULE, CONSISTING OF A RELOCATABLE DECK OF SYSTEM TEN CODE AND DATA, SUITABLE FOR THE LINK EDIT PROGRAM, AND/OR A SYMBOL TABLE, WHICH CONTAINS THE COMPILED FORM OF ALL DATA DEFINITIONS AND DECLARATIONS OF THAT BLOCK PLUS THOSE TO WHICH IT IS SUBORDINATE. NORMAL MODE OF OPERATION IS VIA A SYSTEM TEN OPERATING AS A HASP TERMINAL TO THE 370. THE SYSTEM TEN USES TWO PARTITIONS (FOR SCA AND RJF) FOR THIS FUNCTION. REMAINING PARTITIONS MAY BE USED FOR TEXT EDIT. ONE OR MORE BLOCKS OF SINGLE SOURCE CODE MAY BE SUBMITTED, AS PART OF A TEXT EDITOR FILE, TO BE COMPILED. WHEN THE COMPILATION IS COMPLETE, RESULTS ARE ROUTED BACK TO THE SYSTEM TEN. ANY SYMBOL TABLE OUTPUT IS RETAINED IN A 370 PERMANENT FILE, SO THAT BLOCKS SUBORDINATE TO A PREVIOUSLY COMPILED BLOCK MAY BE SUBMITTED FOR COMPILATION WITHOUT RESUBMITTING THE HIGHER LEVEL BLOCK.

5. OPERATIONAL SOFTWARE

5.1 SYSTEM CONSIDERATIONS

5.1.1 HARDWARE CONSIDERATIONS

REQUIRED HARDWARE IS AN IBM 370 CAPABLE OF RUNNING A 200K JOB UNDER OS/VS2, WITH AT LEAST 400K BYTES OF DISC STORAGE, PLUS

ENOUGH ADDITIONAL DISC SPACE TO HOLD OBJECT FILES. (ONE 3330 DISC PACK SHOULD BE MORE THAN SUFFICIENT). ALSO, A CARD READER, CARD PUNCH AND PRINTER, OR A HASP INTERFACE CONNECTED TO A HASP TERMINAL.

5.1.2 SOFTWARE CONSIDERATIONS

THIS PRODUCT RUNS UNDER OS/VS2. IT REQUIRES THE PL/I LEVEL H COMPILER TO CONVERT THE SOURCE TO AN EXECUTABLE PROGRAM. IF THE HASP CONNECTION IS USED, THEN THE HASP SOFTWARE IS ALSO REQUIRED.
A SEPARATE PROGRAM, CALLED ANALYZER, IS REQUIRED IF THE SYNTAX OF THE SINGLE LANGUAGE IS TO BE MODIFIED. THIS PROGRAM REQUIRES THE SAME HARDWARE ENVIRONMENT AND OS/VS2, EXCEPT THAT 300K JOB SPACE AND 1000K DISC ARE REQUIRED. NOTE THAT THIS JOB NEED NOT EXECUTE ON THE SAME SYSTEM, AS LONG AS SOME MEANS IS AVAILABLE TO PUT THE OUTPUT OF ANALYZER ON FILE AT THE COMPILER SITE.

5.2 EXTERNAL ORGANIZATION

5.2.1 GENERATION

THIS PRODUCT EXISTS AS A SOURCE FILE OF A PL/I PROGRAM. IT IS COMPILED IN ONE COMPLETE COMPILATION. THE COMPILED PROGRAM REQUIRES A FILE OF THE SYNTAX WHICH IS PREPARED BY THE ANALYZER PROGRAM AND CATALOGUED IN THE FILE SYSTEM.

5.2.1.2 CUSTOMIZATION

NO SPECIFIC CUSTOMIZATION FEATURES HAVE BEEN MADE.

5.2.1.3 INSTALLATION

INSTALLATION UNDER ANY SUITABLE OS/VS2 SYSTEM IS ACHIEVED BY CATALOGUING THE COMPILER OBJECT FILE AND THE SYNTAX FILE, PLUS DEFINING A PERMANENT FILE TO BE USED BY THE OBJECT MANAGER FOR RETAINING SYMBOL TABLES. THE OBJECT AND SYNTAX FILES ARE CURRENTLY CATALOGUED AT SISCO IN SUNNYVALE AS SINGLE.XCOM AND SINGLE.SYNTAX.

5.2.2 INVOCATION

5.2.2.1 INVOCATION - OPERATOR INPUTS

SINCE THIS RUNS AS A BATCH JOB UNDER OS/VS2, NO OPERATOR INPUTS ARE REQUIRED.

5.2.2.2 INVOCATION - OPERATOR OUTPUTS

STANDARD OS/VS2 OPERATOR INFORMATION.

5.2.2.3 INVOCATION - PROGRAMMER INPUTS

IF RUNNING LOCALLY, THE PROGRAMMER MAY PUT HIS JOB DECK IN THE PUBLIC CARD READER.
IF RUNNING VIA HASP AND SYSTEM TEN RJF, HE SUBMITS AN RJF DISC CONTROL CARD (SEE RJF DISC MANUAL) TO CAUSE HIS FILE, WHICH MUST BE THE IMAGE OF A JOB DECK, TO BE TRANSMITTED FOR EXECUTION. ALTERNATIVELY, HE MAY SUBMIT THE JOB DECK VIA THE SYSTEM TEN CARD READER.
THE JOB DECK WILL CONTAIN ONE OR MORE BLOCKS OF SINGLE SOURCE CODE. PRIOR TO SUBMISSION OF THE JOB DECK, EACH OF THESE BLOCKS MUST BE EDITED WITH THE COMMAND

USE.SINGLE.RJSHIP

THIS PROCEDURE CONVERTS ALL '\' AND '↑' CHARACTERS TO '&' AND

'%' RESPECTIVELY. THE COMPILER EXPECTS THE LATTER CHARACTERS RATHER THAN THE FORMER. THIS PROCESS IS NECESSARY TO AVOID TRANSMISSION ERRORS IF THE JOB IS SUBMITTED REMOTELY, OR SYNTAX ERRORS IF THE JOB IS SUBMITTED LOCALLY. IF THE BLOCKS TO BE TRANSMITTED CONTAIN NO '\' AND/OR '↑' CHARACTERS, THE GLOBAL CHANGES CALLED FOR IN THE USE FILE WILL RETURN CONTROL TO THE TERMINAL WITH THE MESSAGE "FAILS!". IN THIS EVENT, IT IS NECESSARY TO ENTER THE COMMAND

USE.SINGLE.RJSHIP N

WHERE N IS 1+NUMBER OF LINES OF THE USE FILE ALREADY EXECUTED. (THE LINES EXECUTED ARE DISPLAYED ON THE CONSOLE.)

5.2.2.4 INVOCATION - PROGRAMMER OUTPUTS

THE OUTPUTS DUE TO INVOCATION CONSIST OF 3 PAGES OF PRINTER OUTPUT (4 PAGES IF RUNNING REMOTELY). THESE ARE A BEGINNING AND ENDING JOB BANNER PAGE, A HASP LOG (IF HASP IS USED), AND A JCL LIST PAGE USING THE JCL CONTROL MSGLEVEL=(2,0).

5.2.3 PROCESSING

5.2.3.1 PROCESSING - OPERATOR INPUTS

NONE

5.2.3.2 PROCESSING - OPERATOR OUTPUTS

NONE

5.2.3.3 PROCESSING - PROGRAMMER INPUTS

THE PROGRAMMER INPUT CONSISTS OF A JCL JOB CARD, A FURTHER SET OF JCL CARDS MAINTAINED BY THE SINGLE COMPILER PROJECT

AND CATALOGUED AT THE SOFTWARE DEVELOPMENT LAB UNDER THE NAME SINGLE.JCL, AND ANY NUMBER OF COMPILATION BLOCKS. FOR THE SPECIFICS OF A JCL JOB CARD, SEE THE JCL MANUAL.

5.2.3.3.1 SYNTAX

THE SYNTAX OF SINGLE SOURCE PROGRAMS IS GIVEN BELOW. THIS IS THE SYNTAX RECOGNIZED BY THE COMPILER PROPER, BUT SEE SECTION 3.2.2 FOR DETAILS OF THE LOW LEVEL SYNTAX WHICH IS RECOGNIZED BY THE INPUT SCANNER. ONLY THE FORMAL SYNTAX DEFINITIONS ARE GIVEN HERE, WITH NOTES ON THE SEMANTICS OF EACH DEFINITION IN THE FOLLOWING SECTION. APPENDIX B, DEFINITION CROSS-REFERENCES, SHOULD BE OF HELP IN READING THE SYNTAX.
IN THIS AND THE FOLLOWING SECTION, DEFINITIONS MARKED WITH AN ASTERISK ARE SUBJECT TO SPECIAL RESTRICTIONS ON THIS IMPLEMENTATION. TYPICALLY, THEY REPRESENT DEFERRED FEATURES. THE SPECIFIC RESTRICTION IS INDICATED IN THE NEXT SECTION.

```
1     <RUN> ::= <COMPILATION> .!.
2             ! <OPTIONS> <COMPILATION> .!.

3     <OPTIONS> ::= OPTIONS ( <OPT LST> )

4     <OPT LST> ::= <OPT ITEM>
5                 ! <OPT LST> , <OPT ITEM>

6     <OPT ITEM> ::= <IDENTIFIER>
7  *                ! <IDENTIFIER> <SET> <IDENTIFIER>
8  *                ! <IDENTIFIER> <SET> <NUMBER>
9  *  <SET> ::= =

10    <COMPILATION> ::= <CONTEXT NAME> <CONTEXT BLK>
11                    ! <PROG NAME> <PARENT DECL> <CONTEXT BLK>
                        <XEQ STAT LST>

12    <CONTEXT NAME> ::= CONTEXT <IDENTIFIER>

13    <PROG NAME> ::= PROCEDURE <IDENTIFIER>
14 *                ! TASK <IDENTIFIER>

15    <PARENT DECL> ::= PARENT <IDENTIFIER>
16                    ! <EMPTY>

17    <CONTEXT BLK> ::= <DECL LST>
18                    ! <EMPTY>
```

```
19    <DECL LST> ::= <DEF ITEM>
20                 ! <DECL LST> <DEF ITEM>

21    <DEF ITEM> ::= <CHARMAP DEF> END CHARMAP
22                 ! <PARAM DEF> END PARAMETER
23                 ! <TYPE DEF> END TYPE
24                 ! <CONST DECL> END CONSTANT
25                 ! <PRESET DECL> END PRESET
26                 ! <VAR DECL> END VARIABLE
27                 ! <FLAG DECL> END FLAG
28                 ! <SUBROUTINE DEF> END <IDENTIFIER>

29    <PARAM DEF> ::= <PARAMETER> <PARAM DESCRIP>
30                  ! <PARAM DEF> <PARAM DESCRIP>

31    <PARAMETER> ::= PARAMETER

32    <PARAM DESCRIP> ::= <PARAM ID LST> <TCOLON> <EXPRES>

33    <TCOLON> ::= :

34    <PARAM ID LST> ::= <PARAM IDENTIFIER>
35                     ! <PARAM ID LST> , <PARAM IDENTIFIER>

36    <PARAM IDENTIFIER> ::= £ <IDENTIFIER>

37    <TYPE DEF> ::= <TYPE HEAD> <TYPE DESCRIPT>
38                 ! <TYPE DEF> <TYPE DESCRIPT>

39    <TYPE HEAD> ::= TYPE

40    <TYPE DESCRIPT> ::= <TYPE NAME> <RECORD> <VAR DECL LST>
                            END <IDENTIFIER>
41                      ! <TYPE NAME> <LINK> <LINK DECL>
                            END <IDENTIFIER>
42                      ! <TYPE NAME> <SCALAR>

43    <TYPE NAME> ::= <IDENTIFIER> =

44    <RECORD> ::= RECORD

45    <LINK> ::= PROCEDURE LINK
46 *            ! TASK LINK
47 *            ! TASK OVERLAY
48 *            ! PROCEDURE OVERLAY
```

```
49    <LINK DECL> ::= <VAR DECL LST> <ASSIGNS> <VAR DECL LST>
50                  ! <VAR DECL LST>
51                  ! <ASSIGNS> <VAR DECL LST>
52                  ! LIKE <IDENTIFIER>
53                  ! <EMPTY>

54    <ASSIGNS> ::= ASSIGNS

55    <SCALAR> ::= <SCALAR HEAD> <IDENTIFIER LST> )

56    <SCALAR HEAD> ::= (

57    <CHARMAP DEF> ::= CHARMAP

58    <VAR DECL> ::= <VARIABLE> <VAR DECL LST>

59    <VAR DECL LST> ::= <VAR DESCRIP>
60                     ! <VAR DECL LST> <VAR DESCRIP>

61    <VARIABLE> ::= VARIABLE

62    <CONST DECL> ::= <CONSTANT> <CON DECL LST>

63    <CONSTANT> ::= CONSTANT

64    <CON DECL LST> ::= <CON DESCRIP>
65                     ! <CON DECL LST> <CON DESCRIP>

66    <CON DESCRIP> ::= <VAR DESCRIP> <VCOLON> <VALUE LST>

67    <VCOLON> ::= :

68    <PRESET DECL> ::= <PRESET> <CON DECL LST>

69    <PRESET> ::= PRESET

70 *  <FLAG DECL> ::= <FLAG> <FLAG DESCRIP>
71 *                ! <FLAG DECL> <FLAG DESCRIP>

72 *  <FLAG> ::= FLAG

73 *  <FLAG DESCRIP> ::= <IDENTIFIER LST> <CHECK SPEC>

74    <VALUE LST> ::= <VALUE GEN>
75                  ! <VALUE LST> , <VALUE GEN>

76    <VALUE GEN> ::= <EXPRES>
```

```
                    ! <RANGE PART> <RANGE LIST> : <EXPRES>

<RANGE LIST> ::= <RANGE SPEC>
                  ! <RANGE LIST> , <RANGE SPEC>

<RANGE SPEC> ::= <VALUE PART>

<RANGE PART> ::= RANGE <REPLACE>

<VALUE PART> ::= <EXPRES>
                  ! <O TO> <EXPRES> <STEP PART>
                  ! <EXPRES> <TO> <EXPRES> <STEP PART>

<O TO> ::= TO

<TO> ::= TO

<STEP PART> ::= <STEP> <EXPRES>
                ! <EMPTY>

<STEP> ::= STEP

<VAR DESCRIP> ::= <IDENTIFIER LST> <TCOLON> <TYPE> <CHECK SPEC>

<TYPE> ::= <SIMPLE TYPE>
          ! <ARRAY TYPE>

<SIMPLE TYPE> ::= <IDENTIFIER>
                  ! INT <PQUALIF>
                  ! BINT <PQUALIF>
                  ! STRING <PQUALIF>
                  ! INDEX <FAKE ARROW> <IDENTIFIER>
                  ! POWERSET OF <IDENTIFIER>

<FAKE ARROW> ::= - >

<ARRAY TYPE> ::= ARRAY <PQUALIF> <OF> <SIMPLE TYPE>

<PQUALIF> ::= <LEFT SS><EXPRES><RIGHT SS>

<LEFT SS> ::= [

<RIGHT SS> ::= ]

<CHECK SPEC> ::= CHECK <SPEC LST>
                 ! <EMPTY>
```

```
106     <SPEC LST> ::= <SPEC ITEM>
107                  ! <SPEC LST> , <SPEC ITEM>

108     <SPEC ITEM> ::= <IDENTIFIER>

109     <OF> ::= OF

110     <IDENTIFIER LST> ::= <IDENTIFIER>
111                        ! <IDENTIFIER LST> , <IDENTIFIER>

112     <SUBROUTINE DEF> ::= <SUB HEAD> <XEQ STAT LST>

113     <SUB HEAD> ::= SUBROUTINE <IDENTIFIER>

114     <XEQ STAT LST> ::= <XEQ STAT>
115                      ! <XEQ STAT LST> <XEQ STAT>

116     <LAB STAT LST> ::= <GROUP NUMBER> <XEQ STAT>
117                      ! <LAB STAT LST> <GROUP NUMBER> <XEQ STAT>

118     <GROUP NUMBER> ::= <NUMBER> :
119                      ! :

120     <XEQ STAT> ::= <BAL STAT>
121                  ! <UNBAL IF>

122     <BAL STAT> ::= <SIMPLE XEQ>
123                  ! <BLOCK STAT>
124                  ! <CASE STAT>
125                  ! <CYCLE STAT>

126     <SIMPLE XEQ> ::= <REP STAT>
127                    ! <IF STAT>
128                    ! RECYCLE <GROUP NAME> :
129                    ! LEAVE <GROUP NAME> :
130 *                  ! START <IDENTIFIER> <PARAMS>
131 *                  ! START <IDENTIFIER> <PARAMS> <SOR> <BAL STAT>
132 *                  ! DEFER <IDENTIFIER> <PARAMS>
133 *                  ! WAIT <IDENTIFIER> <EVENT>
134 *                  ! SET <IDENTIFIER> <SET TO> <EXPRES>
135 *                  ! LOCK <IDENTIFIER>
136 *                  ! UNLOCK <IDENTIFIER>
137 *                  ! LOCK <IDENTIFIER> <LOR> <BAL STAT>
138 *                  ! UNLOCK <IDENTIFIER> <LOR> <BAL STAT>
139                  ! DO <IDENTIFIER>
140                  ! CALL <IDENTIFIER> <PARAMS>
141                  ! RETURN
```

```
142 *                    ! STOP
143                      ! PASS
144 *                    ! <ENABLE> <EXPRES>
145 *                    ! <DISABLE> <EXPRES>
146 *                    ! TRANSFER <IDENTIFIER> <PARAMS>
147                      ! INP <IO PARAMS>
148                      ! OUT <IO PARAMS>
149                      ! STATUS <STORE REF>
150 *                    ! STATUS <STORE REF> , <STORE REF>

151 *   <ENABLE> ::= ENABLE

152 *   <DISABLE> ::= DISABLE

153 *   <SOR> ::= OR

154 *   <LOR> ::= OR

155 *   <SET TO> ::= TO

156     <IO PARAMS> ::= <STORE REF> <PARM C> <EXPRES> <PARM C>
                        <EXPRES>
157 *                 ! <STORE REF> <PARM C> <EXPRES> <PARM C>
                        <EXPRES> , <STORE REF>

158     <PARM C> ::= ,

159     <PARAMS> ::= ( <STORE REF> )
160                ! <EMPTY>

161 *   <EVENT> ::= <CONDITION>
162 *             ! FINISH <IDENTIFIER>

163     <REP STAT> ::= <STORE LST> <REPLACE> <EXPRES>
164                  ! INCR <STORE LST> <BY PART>
165                  ! DECR <STORE LST> <BY PART>

166     <REPLACE> ::= : =

167     <BY PART> ::= <BY> <EXPRES>
168                 ! <EMPTY>

169     <BY> ::= BY

170     <STORE LST> ::= <STORE REF>
171                   ! <STORE LST> , <STORE REF>
```

```
172     <STORE REF> ::= <VAR REF> <EXPRES PAIR>
173                   ! <STORE REF> , <VAR REF> <EXPRES PAIR>

174     <VAR REF> ::= <VAR>
175                 ! <VAR> <LEFT SS> <EXPRES> <RIGHT SS>

176     <VAR> ::= <IDENTIFIER>
177             ! @

178     <EXPRES PAIR> ::= <EXPRES PAIR (><EXPRES> <EXPRES PAIR C>
                          <EXPRES> )
179                     ! <EXPRES PAIR (> <EXPRES> )
180                     ! <EMPTY>

181     <EXPRES PAIR (> ::= (

182     <EXPRES PAIR C> ::= ,

183     <IF STAT> ::= <IF PART> <BAL STAT> <ELSE> <BAL STAT>

184     <UNBAL IF> ::= <IF PART> <XEQ STAT>
185                  ! <IF PART> <BAL STAT> <ELSE> <UNBAL IF>

186     <IF PART> ::= <IF> <CONDITION> THEN

187     <IF> ::= IF

188     <ELSE> ::= ELSE

189     <CONDITION> ::= <COND TERM>
190                   ! <CONDITION> OR <COND TERM>

191     <COND TERM> ::= <COND FACT>
192                   ! <COND TERM> AND <COND FACT>

193     <COND FACT> ::= <COND PRIM>
194                   ! NOT <COND PRIM>

195     <COND PRIM> ::= <EXPRES> <REL OP> <EXPRES>
196                   ! ( <CONDITION> )

197     <REL OP> ::= <
198                ! >
199                ! =
200                ! <GR OR EQ>
201                ! <LT OR EQ>
202                ! <NOT EQ>
```

```
203     <GR OR EQ> ::= > =

204     <LT OR EQ> ::= < =

205     <NOT EQ> ::= ↑ =

206     <BLOCK STAT> ::= <BLOCK HEAD> <XEQ STAT LST> END
                         <GROUP NAME> :

207     <BLOCK HEAD> ::= BEGIN <GROUP NAME> :

208     <CASE STAT> ::= <CASE> <EXPRES> <CASE OF> <LAB STAT LST>
                        END CASE

209     <CASE> ::= CASE

210     <CASE OF> ::= OF

211     <CYCLE STAT> ::= <CYCLE HEAD><QUALIFIER><XEQ STAT LST> NEXT
                         <GROUP NAME> :

212     <CYCLE HEAD> ::= CYCLE <GROUP NAME>

213     <QUALIFIER> ::= <CONTROL PART> :

214     <GROUP NAME> ::= <IDENTIFIER>
215                    ! <EMPTY>

216     <CONTROL PART> ::= WHILE <CONDITION>
217                      ! <FOR PART> <RANGE LIST>
218                      ! <EMPTY>

219     <FOR PART> ::= FOR <IDENTIFIER> : =

220     <EXPRES> ::= <TERM>
221
222                ! <EXPRES> + <TERM>
223                ! <EXPRES> - <TERM>

224     <TERM> ::= <PRIMARY>
225              ! <TERM> * <PRIMARY>
226              ! <TERM> / <PRIMARY>
227              ! <TERM> \ <PRIMARY>

228     <PRIMARY> ::= <STORE REF>
229                 ! RANGE
```

```
      230                   ! <NUMBER>
      231                   ! <PARAM IDENTIFIER>
      232                   ! <QSTRING>
      233                   ! MIN ( <EXPRES> , <EXPRES> )
      234                   ! MAX ( <EXPRES> , <EXPRES> )
      235                   ! ASCII64 ( <EXPRES> )
      236                   ! CONVIS ( <EXPRES> )
      237                   ! CONVSI ( <EXPRES> )
      238 *                 ! FLAGVAL ( <IDENTIFIER> )
      239                   ! ALL
      240                   ! ABS(<EXPRES>)
      241                   ! EMPTY
      242                   ! ( <EXPRES> )
```

5.2.3.3.2 SEMANTICS

THE FOLLOWING MATERIAL DISCUSSES THE RESTRICTIONS AND REQUIREMENTS OF THE SEMANTICS OF THE COMPILER, THAT IS, THE LOGIC EXECUTED BY THE COMPILER WHEN AN INSTANCE OF A SPECIFIC DEFINITION HAS BEEN RECOGNIZED, AND THE RESTRICTIONS AND REQUIREMENTS, IF ANY, OF THE LOW LEVEL SYNTAX.

1 <RUN> ::= <COMPILATION>←!←
2 ! <OPTIONS><COMPILATION>←!←

THE '←!←' TERMINATOR IS RECOGNIZED BY THE SCANNER ONLY IF IT BEGINS IN COLUMN 1 OF A SOURCE INPUT LINE. THE LAST BLOCK OF THE RUN SHOULD BE TERMINATED BY THE OS TERMINATOR '/*' RATHER THAN THE '←!←'. THIS TERMINATOR MUST ALSO BEGIN IN COLUMN 1 TO BE RECOGNIZED AS SUCH. ON OUTPUT, THE TERMINATOR IS ALWAYS '←!←'.

6 <OPT ITEM> ::= <IDENTIFIER>
7 * ! <IDENTIFIER><SET><IDENTIFIER>
8 * ! <IDENTIFIER><SET><NUMBER>

IN THE CURRENT IMPLEMENTATION, INSTANCES OF 7 AND 8 ARE IGNORED. INSTANCES OF 6 ARE LIMITED TO SIX POSSIBLE CASES: DATA,CODE,CHECK,ALIST,SAVE,MEM1 AND THEIR NEGATIONS: NODATA,NOCODE,NCHECK,NALIST,NOSAVE,MEM2.

DATA CAUSES AN ABSOLUTE DECK OF LOADABLE DATA TO BE PRODUCED. THIS INCLUDES ALL LOADABLE DATA IN HIGHER LEVEL BLOCKS.

CODE CAUSES A RELOCATABLE DECK OF OBJECT CODE TO BE GENERATED.

ALIST CAUSES AN ASSEMBLY-LIKE LISTING OF THE DATA DECLARATIVES IN EFFECT FOR THIS BLOCK, AND IF THE CODE OPTION IS SELECTED, ASSEMBLY-LIKE LISTING OF THE GENERATED OBJECT CODE.

SAVE CAUSES THE COMPILED SYMBOL TABLE TO REPLACE THE PREVIOUS SYMBOL TABLE FOR THAT BLOCK (IF ANY) IN THE OBJECT FILE. NOSAVE WILL CAUSE THE COMPILED SYMBOL TABLE TO NOT BE SAVED.

CHECK PROVIDES FOR COMPLETE CROSS-REFERENCING OF ALL IDENTIFIERS EXCEPT GROUP NAMES. NOCHECK RESULTS IN CROSS-REFERENCING ONLY THOSE DATA ITEMS FOR WHICH CHECK OPTIONS WERE SPECIFIED.

MEM1 AND MEM2 CAUSE THE BLOCK TO BE ALLOCATED IN PARTITION OR COMMON MEMORY RESPECTIVELY.

ANY OTHER OPTION ITEM WILL CAUSE AN ERROR MESSAGE.

DEFAULT OPTIONS ARE NODATA,CODE,NOSAVE,NOCHECK,NALIST,MEM1. AN OPTION, ONCE SPECIFIED, REMAINS IN EFFECT FOR SUBSEQUENT COMPILATION BLOCKS UNLESS OVER-RIDDEN BY ITS NEGATION.

```
10    <COMPILATION> ::= <CONTEXT NAME> <CONTEXT BLK>
11                    ! <PROG NAME> <PARENT DECL> <CONTEXT BLK>
                        <XEQ STAT LST>
```

WHEN A PROGRAM BLOCK IS INVOKED, CONTROL PASSES TO THE FIRST STATEMENT OF THE <XEQ STAT LST>.

```
14 *  <PROG NAME> ::= TASK <IDENTIFIER>
```

IN THE CURRENT IMPLEMENTATION, TASK IS RESTRICTED TO THE HIGHEST LEVEL EXECUTABLE BLOCK, THAT IS, ITS PARENT MUST BE A CONTEXT BLOCK. THE USE OF TASKS IN ASYNCHRONOUS CODING IS A DEFERRED FEATURE.

```
15    <PARENT DECL> ::= PARENT <IDENTIFIER>
16                    ! <EMPTY>
```

IF THE NAMED PARENT IS NOT FOUND ON FILE, AN ERROR MESSAGE IS GENERATED. 16 IS INCLUDED SO THAT THE COMPILER CAN RECOGNIZE THE CONDITION AND FLAG THE ERROR SPECIFICALLY WITHOUT CAUSING A LONG SEQUENCE OF ERROR MESSAGES.

```
17    <CONTEXT BLOCK> ::= <DECL LIST>
```

ALTHOUGH THE SYNTAX SAYS THAT THE DEFINITION ITEMS MAY APPEAR IN ANY ORDER, THE SEMANTICS REQUIRES THEM IN THE ORDER

CHARMAP, PARAMETER, TYPE, CONSTANT, PRESET, VARIABLE, FLAG,
SUBROUTINE. EACH ITEM, OF COURSE, NEED NOT APPEAR.

32 <PARAM DESCRIP> ::= <PARAM ID LIST><TCOLON><EXPRES>

THE RESTRICTIONS ON <EXPRES> FOR THIS DEFINITION ARE:
THE FOLLOWING FORMS OF <PRIMARY> ARE NOT ALLOWED: 'RANGE', 'ALL',
'EMPTY', AND 'FLAGVAL'. <STORE REF> IS ALLOWED AS A PRIMARY
ONLY IF IT REFERS TO A PREVIOUSLY DEFINED CONSTANT OR PRESET
DATA ITEM, OR TO A PREVIOUSLY DEFINED CHARMAP FUNCTION. (SEE
SEMANTICS, RULE 228). <PARAM IDENTIFIER> IS ALLOWED AS A PRIMARY
ONLY IF THAT IDENTIFIER HAS BEEN PREVIOUSLY DEFINED.
ANY INSTANCE OF <EXPRES> WITHIN A PRIMARY IS SUBJECT TO THE SAME
RESTRICTIONS ON PRIMARIES.

THE <EXPRES> MUST EVALUATE TO A NON-NEGATIVE INTEGER OR STRING.

40 <TYPE DESCRIPT> ::= <TYPE NAME> <RECORD> <VAR DECL LST>
END <IDENTIFIER>
41 ! <TYPE NAME> <LINK> <LINK DECL>
END <IDENTIFIER>
42 ! <TYPE NAME> <SCALAR>

43 <TYPE NAME> ::= <IDENTIFIER> =

RULE 40 IS THE FORM FOR DECLARING STRUCTURED DATA ELEMENTS. THE
<IDENTIFIER> IN THE <TYPE NAME> IS DEFINED TO BE A DATA TYPE. A
DATA ITEM MAY THEN BE DECLARED IN THE DECLARATION SECTION AS
HAVING THE TYPE WHOSE NAME AND STRUCTURE ARE DEFINED HERE. THE
ELEMENTS OF THE <VAR DECL LIST> ARE THE NEXT LEVEL "FIELDS" OF
THE RECORD. THESE FIELDS MAY IN TURN BE OF SOME RECORD TYPE,
AND THUS ONE MAY HAVE SUBFIELDS WITHIN FIELDS, ETC.

RULE 41 IS USED TO DEFINE THE BLOCKS WHICH ARE OWNED BY THIS
BLOCK, AND TO DEFINE THE ARGUMENTS, IF ANY, WHICH ARE USED IN
REFERENCING THOSE BLOCKS. THE <TYPE NAME> IS THE NAME OF A
BLOCK AND THE NAME OF A RECORD TYPE WHOSE STRUCTURE IS GIVEN
BY THE <LINK DECL>. WHEN SUCH A BLOCK IS REFERENCED, THE
ARGUMENT MUST BE A RECORD OF THAT TYPE.

IN RULES 40 AND 41, IF THE <IDENTIFIER> AFTER THE 'END' DOES NOT
MATCH THAT IN THE <TYPE NAME> A WARNING MESSAGE RESULTS.

RULE 42 DEFINES THE <TYPE NAME> AS CORRESPONDING TO A SET OF
NAMED VALUES LISTED IN THE <SCALAR>. A DATA ITEM DECLARED
TO BE OF TYPE <TYPE NAME> RANGES OVER THAT SET OF VALUES.

```
45    <LINK> ::= PROCEDURE LINK
46 *           ! TASK LINK
47 *           ! TASK OVERLAY
48 *           ! PROCEDURE OVERLAY

 IN THE CURRENT IMPLEMENTATION, INSTANCES OF 47 AND 48 WILL
 BE INTERPRETED AS IF THEY WERE 46 AND 45 RESPECTIVELY.
 SUPPORT FOR OVERLAYS IS A DEFERRED FEATURE.  FOR THIS IMPLEM-
 ENTATION, AN INSTANCE OF 46 OUTSIDE OF A CONTEXT BLOCK WILL
 CAUSE AN ERROR MESSAGE.


49    <LINK DECL> ::= <VAR DECL LST> <ASSIGNS> <VAR DECL LST>
50                  ! <VAR DECL LST>
51                  ! <ASSIGNS> <VAR DECL LST>
52                  ! LIKE <IDENTIFIER>
53                  ! <EMPTY>

 WHEN THE BLOCK NAMED IN THE LINK DECLARATION IS SUBSEQUENTLY
 COMPILED, ASSIGNMENT OF A VALUE TO A FIELD OF THE ARGUMENT RECORD
 WILL CAUSE AN ERROR UNLESS THAT FIELD NAME FOLLOWS THE 'ASSIGNS'
 WITHIN THE LINK DECLARATION.
 RULE 52 HAS THE EFFECT OF MAKING THIS ARGUMENT RECORD TYPE
 SYNONYMOUS WITH SOME PREVIOUSLY DEFINED ARGUMENT RECORD TYPE, SO
 THAT A RECORD OF THE LATTER TYPE MAY BE USED AS AN ARGUMENT RECORD
 IN INVOKING THIS BLOCK.

57    <CHARMAP DEF>::=CHARMAP

 THE SYNTAX OF THE DEFINITION OF A CHARMAP IS NOT CONVENIENTLY
 EXPRESSIBLE IN A CONTEXT-FREE GRAMMAR, SO THE RECOGNITION OF
 THIS RESERVED WORD CAUSES CONTROL TO PASS TO A SPECIAL SCANNER
 WHICH COMPLETELY ANALYZES THE CHARMAP DEFINITIONS.

 AN INCOMPLETE SYNTAX FOR THE CHARMAP DEFINITION IS:

 <CHARMAP DEF> ::= CHARMAP <MAPDEF LIST>

 <MAPDEF LIST> ::= <MAPDEF>
                 ! <MAPDEF LIST> <MAPDEF>

 <MAPDEF> ::= <IDENTIFIER> = (<MAP LIST>)

 <MAP LIST> ::= <MAP TERM>
              ! <MAP LIST> <MAP TERM>

 A <MAPDEF> ASSOCIATES A NAME <IDENTIFIER> WITH A FUNCTION
```

WHICH MAPS A ONE-CHARACTER ARGUMENT INTO A SMALL NON-NEGATIVE INTEGER. THE MAPPING IS DEFINED BY GIVING A SEQUENCE OF CHARACTER GROUPS. EACH CHARACTER IN A GROUP MAPS INTO THE THE SAME INTEGER VALUE, AND CHARACTERS IN SUCCESSIVE GROUPS MAP INTO SUCCESSIVE INTEGER VALUES, BEGINNING WITH ZERO. IF A CHARACTER APPEARS IN MORE THAN ONE GROUP AN ERROR MESSAGE RESULTS, AND THE SECOND APPEARANCE OF THE CHARACTER IS IGNORED.

THE RECOGNITION OF <MAP TERM> IS BEST DESCRIBED IN WORDS: AFTER SEEING '(' THE FOLLOWING TEXT IS SCANNED. COMMENTS AND DEBUG CONTROLS ARE RECOGNIZED AND TREATED NORMALLY. INSTANCES OF <NUMBER> ARE IGNORED. THE FIRST CHARACTER OF A <MAPTERM> IS THE COLON, ':'. ANY OTHER CONSTRUCT CAUSES AN ERROR, EXCEPT THE APPEARANCE OF THE PARENTHESIS WHICH TERMINATES THIS INSTANCE OF <MAPDEF>. IF NO ERRORS HAVE OCCURRED, THE COMPILER INSERTS A THREE DIGIT NUMBER IMMEDIATELY PRECEDING THE ':', GIVING THE SEQUENCE NUMBER (AND HENCE THE MAP IMAGE) OF THE CHARACTER GROUP WHICH FOLLOWS. THE NUMBERS START WITH ZERO, AND MAY NOT EXCEED 255.
THE CHARACTER GROUP IS RECOGNIZED AS FOLLOWS: THE FIRST NON-BLANK CHARACTER AFTER THE ':' , IF THE LETTER 'O',CAUSES A TEST TO SEE IF IT PLUS THE NEXT FOUR CHARACTERS FORM THE RESERVED WORD 'OTHER'. IF SO, THE CHARACTER GROUP OF THIS <MAP TERM> IS TO BE ALL CHARACTERS NOT SPECIFIED IN SOME OTHER TERM OF THIS <MAP LIST>. IN ALL OTHER CASES, THE FIRST NON-BLANK AFTER THE ':' IS TREATED AS A DELIMITER. ALL CHARACTERS FOLLOWING THAT DELIMITER AND PRECEDING THE NEXT APPEARANCE OF THAT DELIMITER FORM THE CHARACTER GROUP OF THIS <MAP TERM>. THE CHARACTER GROUP MAY BE EMPTY, AS IN THE CASE '//'. THE SEARCH FOR THE DELIMITER IS SATISFIED BY END-OF-LINE, HOWEVER AN ERROR MESSAGE IS GENERATED IN THIS CASE. IF NO NON-BLANK IS FOUND TO THE RIGHT OF THE COLON IN THAT INPUT LINE, THE SEARCH FOR A DELIMITER TERMINATES AND THE CHARACTER GROUP IS TAKEN TO BE EMPTY. ONCE THE CHARACTER GROUP HAS BEEN IDENTIFIED, THE SUBSEQUENT TEXT IS SCANNED AS ABOVE, SEARCHING FOR ANOTHER <MAPTERM> OR THE PARENTHESIS WHICH TERMINATES THE <MAPDEF>. NOTE THAT THIS TERMINATOR IS NOT RECOGNIZED AS SUCH IF IT APPEARS WITHIN A CHARACTER GROUP, SOME OTHER CONSTRUCT, OR A COMMENT.

62 <CONST DECL> ::= <CONSTANT> <CON DECL LST>

ANY EXECUTABLE STATEMENT WHICH ASSIGNS A VALUE TO A CONSTANT DATA ITEM WILL CAUSE AN ERROR MESSAGE.

70 * <FLAG DECL> ::= <FLAG> <FLAG DESCRIP>

71 * ! <FLAG DECL> <FLAG DESCRIP>

72 * <FLAG> ::= FLAG

73 * <FLAG DESCRIP> ::= <IDENTIFIER LST> <CHECK SPEC>

SUPPORT FOR FLAGS IS A DEFERRED FEATURE. INSTANCES OF ANY OF 70 THROUGH 73 WILL BE IGNORED, AND AN ERROR WILL RESULT.

74 <VALUE LST> ::= <VALUE GEN>
75 ! <VALUE LST> , <VALUE GEN>

MORE THAN ONE VALUE IS PERMISSIBLE ONLY IF THE DATA ITEM BEING DECLARED IS AN ARRAY. IN THE EVENT THAT A VALUE IS ASSIGNED TO A SUBSCRIPT WHICH EXCEEDS THE DIMENSION OF THE ARRAY, AN ERROR MESSAGE RESULTS, AND THE ILLEGAL VALUE IS IGNORED. IN A LIST OF VALUES, ASSIGNMENTS ARE TO ARRAY ITEMS IN SUBSCRIPT ORDER, FIRST VALUE CORRESPONDING TO SUBSCRIPT 0, NEXT TO SUBSCRIPT 1, AND SO ON. THIS ORDER MAY BE OVER-RIDDEN THROUGH THE USE OF A "RANGE" CLAUSE (SEE RULE 77). INDIVIDUAL LISTED VALUES FOLLOWING A "RANGE" CLAUSE ARE ASSIGNED TO SUCCESSIVE SUBSCRIPT VALUES BEGINNING WITH 1+(LAST VALUE ASSIGNED TO "RANGE").

76 <VALUE GEN> ::= <EXPRES>

THE LIMITATIONS ON <EXPRES> FOR THIS DEFINITION ARE: THE FOLLOWING FORMS OF <PRIMARY> ARE NOT ALLOWED: 'RANGE','ALL', 'EMPTY', AND 'FLAGVAL'. <STORE REF> IS ALLOWED AS A PRIMARY ONLY IF IT REFERS TO A PREVIOUSLY DEFINED CONSTANT OR PRESET DATA ITEM, OR TO A PREVIOUSLY DEFINED CHARMAP FUNCTION. (SEE SEMANTICS, RULE 228). <PARAM IDENTIFIER> IS ALLOWED AS A PRIMARY ONLY IF THAT IDENTIFIER HAS BEEN PREVIOUSLY DEFINED. ANY INSTANCE OF <EXPRES> WITHIN A PRIMARY IS SUBJECT TO THE SAME RESTRICTIONS ON PRIMARIES.

THE <EXPRES> MUST EVALUATE TO AN INTEGER OR STRING VALUE. IN ADDITION, THE TYPE OF THE <EXPRES> MUST BE COMPATIBLE WITH THE TYPE OF THE DATA ITEM BEING GIVEN A VALUE (SEE SEMANTICS, RULES 163-5), WHICH HENCE MAY NOT BE OF RECORD, POWERSET OR SCALAR TYPE. IF THE RESULTING VALUE OF THE EXPRESSION EXCEEDS THE SIZE SPECIFIED BY THE <PQUALIF> (SEE RULES 66,90-101), AN ERROR MESSAGE RESULTS, OR A WARNING MESSAGE IN THE CASE OF STRING DATA, WHICH IS TRUNCATED ON THE RIGHT TO FIT THE DECLARED SIZE.

77 ! <RANGE PART> <RANGE LIST> : <EXPRES>

LIMITATIONS ON <EXPRES> FOR THIS DEFINITION ARE THE SAME AS THOSE FOR 76 EXCEPT THAT 'RANGE' IS ALSO AN ADMISSIBLE PRIMARY. THE MEANING OF "RANGE" IS THAT ITS VALUE SPECIFIES A SUBSCRIPT VALUE FOR THE ARRAY, TO WHICH THAT VALUE OF THE <EXPRES> IS TO BE ASSIGNED. IF A MORE THAN ONE ASSIGNMENT IS MADE TO THE SAME SUBSCRIPT VALUE, AN ERROR IS GENERATED AND THE LAST SPECIFIED ASSIGNMENT IS MADE.

```
82    <VALUE PART> ::= <EXPRES>
83                   ! <0 TO> <EXPRES> <STEP PART>
84                   ! <EXPRES> <TO> <EXPRES> <STEP PART>

85    <0 TO> ::= TO

86    <TO> ::= TO

87    <STEP PART> ::= <STEP> <EXPRES>
88                  ! <EMPTY>
```

THE LIMITATIONS ON <EXPRES> FOR THESE DEFINITIONS DEPEND UPON WHERE IN THE SOURCE CODE THEY APPEAR. IN ANY CASE, THE EXPRESSION MUST EVALUATE TO A NON-NEGATIVE INTEGER, AND MAY NOT CONTAIN ANY OF THE PRIMARIES <NUMBER>,<STRING>,'RANGE','ALL','EMPTY' OR 'FLAGVAL'. <PARAM IDENTIFIER> IS ALLOWED ONLY IF THAT PARAMETER HAS PREVIOUSLY BEEN ASSIGNED A VALUE. WITHIN THE DECLARATION SECTION, THE EXPRESSION MUST BE FULLY EVALUATABLE AT COMPILE TIME, AND HENCE ANY <STORE REF> MUST BE TO A PREVIOUSLY DECLARED PRESET OR CONSTANT DATA ITEM. AS USUAL, ANY INSTANCES OF <EXPRES> WITHIN A PRIMARY ARE SUBJECT TO THE SAME RESTRICTIONS ON PRIMARIES AS THE CONTAINING EXPRESSION.
IN 82 ONE VALUE IS GENERATED, THAT GIVEN BY THE EXPRESSION.
IN 83 AND 84 A SEQUENCE OF VALUES IS GENERATED, BEGINNING WITH ZERO IN THE CASE OF 83, OR THE VALUE OF THE FIRST <EXPRES> IN 84. SUCCESSIVE VALUES ARE DERIVED FROM THE PREVIOUS VALUE BY ADDING TO IT THE VALUE OF THE <EXPRES> IN CASE 87, OR THE VALUE 1 IN CASE 88 UNTIL A VALUE GREATER THAN THAT OF THE <EXPRES> PRECEDING THE <STEP PART> IS OBTAINED. THIS VALUE IS DISCARDED AND THE SEQUENCE IS COMPLETE.

```
90    <VAR DESCRIP> ::= <IDENTIFIER LST> <TCOLON> <TYPE> <CHECK SPEC>
```

EACH OF THE IDENTIFIERS IN THE <IDENTIFIER LST> IS DECLARED TO BE A DATA ITEM OF THE INDICATED TYPE.

```
93    <SIMPLE TYPE> ::= <IDENTIFIER>
```

<IDENTIFIER> MUST BE DEFINED AS A TYPE IN THIS OR A HIGHER LEVEL BLOCK.

94 ! INT <PQUALIF>

A VARIABLE OF TYPE INT IS AN INTEGER WHOSE RANGE OF POSSIBLE VALUES IS 0 TO N, WHERE N IS THE VALUE OF THE <PQUALIF>. IF A VALUE OUTSIDE THIS RANGE IS ASSIGNED TO THE VARIABLE, THE RUN-TIME RESULTS ARE UNDEFINED. IF AN OUT-OF-RANGE VALUE IS ASSIGNED TO SUCH AN ITEM AT COMPILE TIME (WITHIN A PRESET OR CONSTANT DECLARATION) AN ERROR MESSAGE RESULTS.
THE MAXIMUM ALLOWED VALUE OF N IS 9,999,999,999. LARGER VALUES WILL CAUSE AN ERROR MESSAGE.

95 ! SINT <PQUALIF>

A VARIABLE OF TYPE SINT IS LIKE A VARIABLE OF TYPE INT WITH AN ALGEBRAIC SIGN. ITS RANGE OF POSSIBLE VALUES IS FROM -N TO N. ASSIGNMENTS OF VALUES OUTSIDE THIS RANGE CAUSE AN ERROR MESSAGE IF MADE AT COMPILE TIME, OR UNDEFINED RESULTS IF MADE AT RUN TIME.
THE MAXIMUM ALLOWED VALUE OF N IS 9,999,999,999. LARGER VALUES WILL CAUSE AN ERROR MESSAGE.

96 ! STRING <PQUALIF>

A DATA ITEM OF TYPE STRING IS A SEQUENCE OF N CHARACTERS OF THE SINGLE CHARACTER SET, WHERE N IS THE VALUE OF <PQUALIF>. ASSIGNMENT TO SUCH A VARIABLE OF FEWER THAN N CHARACTERS CAUSES THOSE CHARACTERS TO BE LEFT ADJUSTED IN THE VARIABLE, WITH BLANKS ADDED ON THE RIGHT TO FILL. ASSIGNMENT OF MORE THAN N CHARACTERS CAUSES ONLY THE LEFT-MOST N CHARACTERS TO BE PLACED IN THE VARIABLE; IF THE ASSIGNMENT IS MADE AT COMPILE TIME A WARNING MESSAGE RESULTS.
THE MAXIMUM ALLOWED VALUE OF N IS 100. LARGER VALUES WILL CAUSE AN ERROR MESSAGE.

97 ! INDEX <FAKE ARROW> <IDENTIFIER>

A DATA ITEM OF TYPE INDEX => <IDENTIFIER> IS PERMISSIBLE ONLY IF THE <IDENTIFIER> IS DECLARED IN THIS OR A HIGHER LEVEL BLOCK AS AN ARRAY <PQUALIF> OF SOME TYPE. THE INDEX ITEM IS EQUIVALENT TO AN ITEM OF TYPE INT <PQUALIF>, EXCEPT THAT ARRAY REFERENCES WILL BE EXECUTED MORE EFFICIENTLY USING THE INDEX ITEM THAN USING AN INT TYPE ITEM AS THE SUBSCRIPT.

98 ! POWERSET OF <IDENTIFIER>

AN ITEM OF TYPE POWERSET IS PERMISSIBLE IF THE <IDENTIFIER> IS DEFINED AS A SCALAR TYPE IN THIS OR A HIGHER LEVEL BLOCK. THE MAXIMUM NUMBER OF COMPONENTS OF THE SCALAR TYPE IS 16. IF THIS IS EXCEEDED AN ERROR RESULTS. A POWERSET DATA ITEM HAS VALUES WHICH ARE SUBSETS OF THE SET OF ALL POSSIBLE VALUES OF THE SCALAR TYPE. THUS IF THE SCALAR HAS N VALUES THE POWERSET REQUIRES N BINARY DIGITS TO REPRESENT ITS VALUES.

100 <ARRAY TYPE> ::= ARRAY<PQUALIF><OF><SIMPLE TYPE>

THE VALUE,N, OF <PQUALIF> INDICATES THAT THE ARRAY CONTAINS N+1 ITEMS, EACH OF THE SAME <SIMPLE TYPE>. THE INDIVIDUAL ITEMS ARE REFERENCED BY SUBSCRIPTING WITH VALUES FROM 0 TO N. USE OF OUT-OF-RANGE SUBSCRIPTS AT RUN TIME CAUSES UNDEFINED RESULTS.

101 <PQUALIF> ::= <LEFT SS> <EXPRES> <RIGHT SS>

THE LIMITS ON <EXPRES> FOR THIS DEFINITION ARE THE SAME AS THOSE FOR 82:87, WITHIN THE DEFINITIONS AND DECLARATIONS.

108 <SPEC ITEM> ::= <IDENTIFIER>

THE ONLY ADMISSIBLE VALUES OF IDENTIFIER ARE F,S,L, AND U. THESE REPRESENT COMPILE TIME CROSS-REFERENCING OF ACTIONS WHICH CAUSE A FETCH, STORE, LOCK, OR UNLOCK OF THE DATA ITEM. L AND U ARE MEANINGFUL ONLY FOR FLAG ITEMS, AND ARE NOT SUPPORTED IN THE CURRENT IMPLEMENTATION.

112 <SUBROUTINE DEF> ::= <SUB HEAD> <XEQ STAT LST>

ANY SUBROUTINE DEFINITIONS IMMEDIATELY PRECEDE THE ACTIONS OF THE BLOCK. NOTE THAT THESE SUBROUTINES HAVE NO ARGUMENTS. THEY ARE INVOKED BY THE 'DO' STATEMENT (SEE 139). CONTROL RETURNS TO THE ACTION FOLLOWING THE 'DO' WHEN THE SUBROUTINE EXECUTES A 'RETURN' STATEMENT OR COMPLETES THE ACTION IMMEDIATELY PRECEDING THE 'END' WHICH TERMINATES ITS DEFINITION. THE DEFINITION OF A SUBROUTINE HAS EFFECT ONLY WITHIN THE BLOCK IN WHICH IT IS DEFINED, AND THE SUBROUTINE CANNOT BE REFERENCED FROM ANY OTHER BLOCK.

114 <XEQ STAT LST> ::= <XEQ STAT>
115 ! <XEQ STAT LST> <XEQ STAT>

WHEN CONTROL IS PASSED TO AN <XEQ STAT LST>, EXECUTION BEGINS WITH THE FIRST STATEMENT. CONTROL FLOWS FROM ONE STATEMENT TO

THE NEXT IN THE LIST UNLESS EXECUTING A STATEMENT CAUSES CONTROL TO BE TRANSFERRED IN SOME OTHER MANNER.

IN ALL EXECUTABLE STATEMENTS, THE RESTRICTIONS ON <EXPRES> ARE THAT <NUMBER>,'RANGE',AND <QSTRING> ARE NOT ALLOWED AS PRIMARIES, AND A SIMILAR RESTRICTION APPLIES TO ANY INSTANCE OF <EXPRES> WITHIN AN ADMISSIBLE <PRIMARY>. FOR OTHER GENERAL RULES ON EXPRESSIONS SEE THE SEMANTICS FOR RULES 220 ET. SEQ. CERTAIN INSTANCES OF <EXPRES> WITHIN EXECUTABLE STATEMENTS MAY HAVE ADDITIONAL RESTRICTIONS, IN WHICH CASE THE SEMANTICS FOR THAT CONSTRUCT CITE THE ADDITIONAL RESTRICTIONS.

118 <GROUP NUMBER> ::= <NUMBER> :
119 ! :

IF THE VALUE OF THE <EXPRES> IN THE CASE STATEMENT IS N, THEN STATEMENT WITH GROUP NUMBER N IS THE ONE WHICH IS EXECUTED. (SEE 208, 116:117). ON INPUT, A NUMBER IS IGNORED. A NUMBER IS CALCULATED FROM POSITION IN THE <LAB STAT LST> AND THIS NUMBER IS INSERTED IN THE OUTPUT. THE MAXIMUM PERMISSIBLE GROUP NUMBER IS 999.

128 (<SIMPLE XEQ>) ! RECYCLE <GROUP NAME> :

IF THE <GROUP NAME> IS EMPTY, CONTROL PASSES TO THE NEAREST PRECEDING 'CYCLE' STATEMENT. OTHERWISE CONTROL PASSES TO THE NEAREST PRECEDING 'CYCLE' STATEMENT WITH THAT <GROUP NAME>. IF THE INDICATED 'CYCLE' DOES NOT EXIST AN ERROR RESULTS.

129 ! LEAVE <GROUP NAME> :

IF <GROUP NAME> IS EMPTY, CONTROL PASSES TO THE FIRST ACTION PAST THE NEAREST FOLLOWING 'END' OR 'NEXT'. OTHERWISE CONTROL PASSES TO THE FIRST ACTION PAST THE 'END' OR 'NEXT' WHICH TERMINATES THE 'BEGIN' OR 'CYCLE' HAVING THAT <GROUP NAME>. IF THE INDICATED POINT DOES NOT EXIST AN ERROR RESULTS.

130 * ! START <IDENTIFIER> <PARAMS>
131 * ! START <IDENTIFIER> <PARAMS> <SOR> <BAL STAT>
132 * ! DEFER <IDENTIFIER> <PARAMS>
133 * ! WAIT <IDENTIFIER> <EVENT>
134 * ! SET <IDENTIFIER> <SET TO> <EXPRES>
135 * ! LOCK <IDENTIFIER>
136 * ! UNLOCK <IDENTIFIER>
137 * ! LOCK <IDENTIFIER> <LOR> <BAL STAT>
138 * ! UNLOCK <IDENTIFIER> <LOR> BAL STAT>

ALL OF THE ABOVE ARE IGNORED IF FOUND. AN ERROR RESULTS. THESE ALL REPRESENT FEATURES NOT SUPPORTED IN THE CURRENT IMPLEMENTATION.

139 ! DO <IDENTIFIER>

EXECUTING THIS STATEMENT CAUSES CONTROL TO PASS TO THE SUBROUTINE WITH THAT <IDENTIFIER>. IF SUCH A SUBROUTINE HAS NOT BEEN DEFINED WITHIN THAT BLOCK AN ERROR MESSAGE RESULTS.

140 ! CALL <IDENTIFIER> <PARAMS>

EXECUTING THIS STATEMENT CAUSES CONTROL TO PASS TO THE PROCEDURE HAVING THAT <IDENTIFIER>. THAT PROCEDURE MUST BE DIRECTLY SUBORDINATE TO A BLOCK WHOSE SCOPE INCLUDES THE CALLING PROGRAM. THE CALLING PROGRAM MAY NOT BE THE PROCEDURE CALLED NOR SUBORDINATE TO IT. VIOLATION OF THESE RULES CAUSES AN ERROR MESSAGE.

141 ! RETURN

EXECUTING THIS STATEMENT CAUSES CONTROL TO RETURN TO THE PROGRAM WHICH INVOKED THIS PROGRAM OR SUBROUTINE. OCCURRENCE OF THIS STATEMENT IN A TASK BLOCK CAUSES AN ERROR.

142 * ! STOP

EXECUTING THIS STATEMENT IS POSSIBLE ONLY FOR A TASK, AND CAUSES TERMINATION OF THAT TASK. UNDER THE CURRENT IMPLEMENTATION THE ONLY TASK IS THE HIGHEST LEVEL PROGRAM. OCCURRENCE OF A 'STOP' WITHIN A PROCEDURE CAUSES AN ERROR.

143 ! PASS

THIS EXECUTABLE STATEMENT PERFORMS NO ACTION. CONTROL FLOWS DIRECTLY THROUGH IT. IT IS USED WHERE THE SYNTAX CALLS FOR AN EXECUTABLE STATEMENT BUT NO ACTION IS DESIRED AT THAT POINT, AS MIGHT HAPPEN IN A 'CASE' STATEMENT.

144 * ! <ENABLE> <EXPRES>
145 * ! <DISABLE> <EXPRES>
146 * ! TRANSFER <IDENTIFIER> <PARAMS>

THESE ARE NOT SUPPORTED IN THE CURRENT IMPLEMENTATION, AND WILL CAUSE AN ERROR.

147 ! INP <IO PARAMS>

148 ! OUT <IO PARAMS>

THESE STATEMENTS MAP INTO SYSTEM TEN IN AND OUT INSTRUCTIONS.

149 ! STATUS <STORE REF>
150 * ! STATUS <STORE REF> , <STORE REF>

ONLY DEFINITION 149 IS ALLOWED IN THE CURRENT IMPLEMENTATION. THE FORM OF 150 IS EXPECTED TO BE USED FOR SYSTEM ELEVEN. THE STATUS STATEMENT CREATES A NUMERIC VALUE FOR THE CURRENT SYSTEM TEN CONDITION CODE, AND ASSIGNS THAT VALUE TO THE <STORE REF>, WHICH MUST BE OF TYPE INT OR SINT. THE VALUES ARE DETERMINED AS FOLLOWS:

STATUS VALUE	CONDITION CODE (AFTER PREVIOUS INSTRUCTION)
0	1
1	2
2	3
3	1,4
4	2,4
5	3,4

156 <IO PARAMS> ::= <STORE REF> <PARM C> <EXPRES> <PARM C> <EXPRES>
157 * ! <STORE REF> <PARM C> <EXPRES> <PARM C> <EXPRES> , <STORE REF>

DEFINITION 157 IS NOT ALLOWED IN THE CURRENT IMPLEMENTATION. IT IS EXPECTED TO BE USED FOR SYSTEM ELEVEN. IN DEFINITION 156, THE <STORE REF> MUST BE TO A STRING OR SUBSTRING OF LENGTH AT MOST 100. THE FIRST <EXPRES> MUST EVALUATE TO A NON-NEGATIVE INTEGER NO GREATER THAN 9. THE SECOND <EXPRES> MUST EVALUATE TO A NON-NEGATIVE INTEGER NO GREATER THAN 7. THE STATEMENTS

```
INP A,B,C
OUT A,B,C
```

GENERATE INSTRUCTIONS EQUIVALENT TO (ASSEMBLER II FORM)

```
R X(Y),L:X(Z)
W X(Y),L:X(Z)
```

WHERE X IS THE ADDRESS OF THE STRING OR SUBSTRING AND L:X IS ITS LENGTH. Y IS THE VALUE OF THE EXPRESSION B, AND Z IS THE VALUE OF THE EXPRESSION C.

```
159     <PARAMS> ::= ( <STORE REF> )
160                ! <EMPTY>

  THE <STORE REF>, IF PRESENT, MUST BE TO A DATA ITEM WHOSE
  TYPE IS <IDENTIFIER>, WHERE THE NAME OF THE PROGRAM INVOKED
  IS ALSO <IDENTIFIER>.  IF NOT, AN ERROR MESSAGE RESULTS.
  THE 'LINK' DEFINITION FOR THE PROGRAM ALSO CAUSES A 'RECORD'
  TYPE OF THE SAME NAME TO BE DECLARED, SO THAT ARGUMENTS MAY
  BE PASSED TO THE PROGRAM BY REFERENCING A DATA RECORD OF THAT
  TYPE.

161 *   <EVENT> ::= <CONDITION>
162 *             ! FINISH <IDENTIFIER>

  THESE ARE NOT SUPPORTED IN THE CURRENT IMPLEMENTATION.

163     <REP STAT> ::= <STORE LST> <REPLACE> <EXPRES>
164                  ! INCR <STORE LST> <BY PART>
165                  ! DECR <STORE LST> <BY PART>

  THESE ARE THE MAIN ACTIONS WHICH ASSIGN VALUES TO DATA ITEMS
  AT RUN TIME.  ALL SUCH ASSIGNMENTS ARE CHECKED AT COMPILE TIME
  FOR COMPATIBILITY OF TYPE, THAT IS, THE TYPE OF THE SOURCE
  ITEM MUST BE COMPATIBLE WITH THAT OF THE RECEIVING ITEM, OR A
  WARNING OR ERROR MESSAGE RESULTS.  COMPATIBLE TYPES FOR A GIVEN
  RECEIVING ITEM TYPE ARE GIVEN BELOW.  IN THIS TABULATION, N
  AND M STAND FOR VALUES OF INSTANCES OF <PQUALIF>, WITH M<=N.

  RECEIVING TYPE   COMPATIBLE TYPES

    INT[N]           INT[M], INDEX=>A  (A:ARRAY[M] OF ...)

    INDEX=>(ARRAY[N])
                     INT[M], INDEX=>(ARRAY[M])

    SINT[N]          INT[M], SINT[M], INDEX=>(ARRAY[M])

    STRING[N]        STRING[M]  (IN THIS INSTANCE, IF M>N AT
                                 RUN TIME THE RESULT IS DEFINED:
                       THE RECEIVING STRING IS FILLED TO ITS
                       CAPACITY WITH CHARACTERS OF THE SOURCE
                       STRING STARTING FROM THE LEFT-MOST
                       POSITION.  IF M < N, THE RESULT IS AS IF
                       THE SOURCE STRING WERE EXTENDED ON THE
                       RIGHT BY N-M BLANK CHARACTERS.

    SCALAR W         SCALAR W
```

POWERSET OF X	POWERSET OF x, SCALAR X (IN THE LATTER CASE THE SCALAR IS CONVERTED TO A POWERSET CONSISTING OF THAT ONE SCALAR VALUE)
ARRAY[N] OF Y	ARRAY[N] OF Z (TYPE OF Z COMPATIBLE WITH THAT OF Y).
R (A <RECORD> TYPE)	R

IF THE TYPES ARE INCOMPATIBLE ONLY IN THAT M>N, THE COMPILER GENERATES A WARNING MESSAGE, AND ASSUMES THAT THE PROGRAM CAN GUARANTEE THAT THE SOURCE ITEM IS IN FACT WITHIN THE RANGE OF THE RECEIVING ITEM. IF THE SOURCE ITEM IS NOT WITHIN THE RANGE OF THE RECEIVING ITEM AT RUN TIME, RESULTS ARE UNDEFINED EXCEPT IN THE CASE OF STRING ITEMS NOTED ABOVE.

IN THE CASE OF STRING ASSIGNMENTS, IF THE SOURCE AND RECEIVING STRING OVERLAP BY VIRTUE OF CONTAINING COMMON POSITIONS WITHIN A LARGER STRING, THE RESULTS OF THE ASSIGNMENT ARE UNDEFINED.

THE INCR AND DECR FORMS ARE VALID ONLY FOR ITEMS OF NUMERIC TYPE (INT, SINT, OR INDEX), AND CAUSE THE DATA ITEM TO BE INCREMENTED OR DECREMENTED RESPECTIVELY BY THE VALUE OF <BY PART>.

```
167     <BY PART> ::= <BY> <EXPRES>
168                 ! <EMPTY>
```

THE <EXPRES> IS THE VALUE BY WHICH THE ITEM IS TO BE INCREMENTED OR DECREMENTED. IN THE SECOND OF THESE CASES, A VALUE OF 1 IS IMPLIED.

```
170     <STORE LST> ::= <STORE REF>
171                   ! <STORE LST> , <STORE REF>
```

THE APPEARANCE OF A LIST OF ITEMS AS RECEIVING ITEMS MEANS THAT THE SAME VALUE IS ASSIGNED TO EACH OF THE RECEIVING ITEMS.

```
174     <VAR REF> ::= <VAR>
175                 ! <VAR> <LEFT SS> <EXPRES> <RIGHT SS>
```

THE SECOND OF THESE IS THE FORM FOR REFERENCING ELEMENTS OF ARRAYS. THE <VAR> MUST NAME AN ARRAY TYPE DATA ITEM, AND THE <EXPRES> MUST BE OF TYPE INT OR INDEX. OUT-OF-RANGE VALUES OF THE EXPRESSION WILL YIELD UNDEFINED RESULTS.

```
176     <VAR> ::= <IDENTIFIER>
177             ! @

  THE SECOND OF THESE IS THE FORM FOR REFERENCING THE RECORD
  WHICH WAS SPECIFIED AS THE ARGUMENT FOR THIS BLOCK WHEN IT WAS
  INVOKED.  IF THE LINK DECLARATION OF THE PARENT BLOCK DID NOT
  SPECIFY ARGUMENTS FOR THIS BLOCK, AN ERROR RESULTS.

178     <EXPRES PAIR> ::= <EXPRES PAIR (><EXPRES> <EXPRES PAIR C>
                          <EXPRES> )
179                     ! <EXPRES PAIR (> <EXPRES> )
180                     ! <EMPTY>

  THE FIRST TWO OF THESE ARE THE FORMS FOR SUBSTRING REFERENCES.
  THE <VAR REF> MUST NAME AN ITEM OF TYPE STRING.  THE TWO
  EXPRESSIONS MUST BE OF TYPE INT OR INDEX.  THEIR VALUES ARE
  INTERPRETED AS THE POSITION WITHIN THE STRING OF THE LEFTMOST
  CHARACTER OF THE SUBSTRING, AND THE NUMBER OF CHARACTERS IN THE
  SUBSTRING, RESPECTIVELY.  POSITION STARTS WITH ZERO AS THE LEFT-
  MOST CHARACTER.  IN THE SECOND FORM A VALUE OF 1 IS TAKEN FOR THE
  LENGTH OF THE SUBSTRING.
  THE SECOND FORM IS ALSO USED FOR CHARMAP REFERENCES.  IF THE
  <VAR REF> NAMES A CHARMAP IDENTIFIER, THE ONE EXPRESSION WITHIN
  THE PARENTHESES MUST EVALUATE TO A STRING OF LENGTH 1.  IN THIS
  INSTANCE, THE <STORE REF> MAY NOT BE TO THE LEFT OF AN ASSIGNMENT.
  THE MEANING OF THE <STORE REF> IS THE INTEGER VALUE INTO WHICH
  THE ONE-CHARACTER ARGUMENT IS MAPPED UNDER THE NAMED CHARMAP.
  IF THE ARGUMENT HAS LENGTH GREATER THAN ONE, THE LEFT-MOST
  CHARACTER OF THE STRING IS TAKEN AS THE ARGUMENT, AND IF THE LENGTH
  WAS EVALUATED AT COMPILE TIME A WARNING MESSAGE RESULTS.
  IF THE CHARMAP HAS NO IMAGE DEFINED FOR THE ARGUMENT CHARACTER, AN
  ERROR RESULTS IF THE CHARMAP IS EVALUATED AT COMPILE TIME, AND
  UNDEFINED RESULTS OCCUR IF IT IS EVALUATED AT RUN TIME.

183     <IF STAT> ::= <IF PART> <BAL STAT> <ELSE> <BAL STAT>

184     <UNBAL IF> ::= <IF PART> <XEQ STAT>
185                  ! <IF PART> <BAL STAT> <ELSE> <UNBAL IF>

  IN EXECUTING EACH OF THESE STATEMENTS, THE CONDITION IN THE
  <IF PART> IS TESTED.  IF THE CONDITION IS TRUE, THE STATEMENT
  FOLLOWING THE <IF PART> IS EXECUTED.  IF IT DOES NOT CAUSE A
  TRANSFER OF CONTROL, THEN UPON COMPLETION OF EXECUTION OF THAT
  STATEMENT, CONTROL PASSES TO THE NEXT STATEMENT AFTER THE ENTIRE
  <IF STAT> OR <UNBAL IF>.  FOR 183 AND 185, IF THE CONDITION IS
  FALSE, THE STATEMENT AFTER THE <ELSE> IS EXECUTED, AND CONTROL
```

PASSES TO THE NEXT STATEMENT UNLESS ALTERED BY THE EXECUTED STATEMENT. 184 IS EQUIVALENT TO <IF PART><XEQ STAT><ELSE> PASS.

IF THE CONDITION IS EVALUATABLE AT COMPILE TIME AND THE TWO EXPRESSIONS CONTAIN NO PRESETS, THEN THE SELECTION OF STATEMENTS TO BE EXECUTED IS MADE AT COMPILE TIME.

195 <COND PRIM> ::= <EXPRES> <REL OP> <EXPRES>

THE COMPILER VERIFIES THAT THE TWO EXPRESSIONS ARE OF COMPARABLE TYPE, OR ELSE GENERATES AN ERROR.
TWO EXPRESSIONS ARE OF COMPARABLE TYPE IF EACH IS

NUMERIC (INT, SINT, OR INDEX),
STRING,
POWERSET OF X,
SCALAR Y,

OR IF ONE IS OF TYPE SCALAR X AND THE OTHER IS OF TYPE POWERSET OF X, IN WHICH CASE THE TYPE SCALAR ITEM IS INTERPRETED AS A POWERSET ITEM (THE SET CONSISTING OF ONLY THAT SCALAR VALUE).

197 <REL OP> ::= <
198 ! >
199 ! =
200 ! <GR OR EQ>
201 ! <LT OR EQ>
202 ! <NOT EQ>

IN THE CASE OF NUMERIC COMPARISONS THE OPERATORS HAVE THE OBVIOUS MEANINGS. IN THE CASE OF STRING COMPARISONS THE NORMAL CONVENTIONS APPLY: THE SHORTER OF THE TWO STRINGS IS CONCEPTUALLY LENTHENED TO EQUAL THE LENGTH OF THE LONGER BY APPENDING BLANK CHARACTERS ON THE RIGHT. THE MEANING OF THE OPERATORS FOR STRING RELATIONS FOLLOWS THE ASCII COLLATING SEQUENCE APPLIED FROM LEFT TO RIGHT.
FOR SCALAR RELATIONS ONLY TWO OPERATORS ARE PERMITTED, EQUAL AND NOT-EQUAL. FOR POWERSET RELATIONS, THE INTERPRETATIONS OF THE OPERATORS CONFORMS TO SET-THEORETIC NOTATION, THAT IS THEY REPRESENT THE SET RELATIONS

PROPERLY CONTAINED,
PROPERLY CONTAINS,
EQUAL,
CONTAINS,
CONTAINED,
NOT EQUAL,

RESPECTIVELY.

206 <BLOCK STAT> ::= <BLOCK HEAD> <XEQ STAT LST> END
<GROUP NAME> :

THE BLOCK STATEMENT ALLOWS WHAT WOULD OTHERWISE BE AN <XEQ STAT LST> TO BE PERCEIVED BY THE SYNTAX PARSER AS ONE STATEMENT, SO THAT MULTIPLE ACTIONS MAY BE PERFORMED, FOR EXAMPLE, ON THE 'ELSE' BRANCH OF AN 'IF' STATEMENT.

208 <CASE STAT> ::= <CASE> <EXPRES> <CASE OF> <LAB STAT LST>
END CASE

THE TYPE OF <EXPRES> MUST BE NUMERIC. IF AT RUN TIME THE VALUE OF <EXPRES> IS LESS THAN ZERO OR GREATER THAN THE NUMBER OF STATEMENTS IN THE <LAB STAT LST>, UNDEFINED RESULTS WILL OCCUR. THE VALUE OF <EXPRES> IS USED TO SELECT THE PARTICULAR STATEMENT WITHIN THE LIST WHICH IS TO BE EXECUTED. UNLESS THAT STATEMENT CAUSES A CONTROL TRANSFER, UPON COMPLETION OF THAT STATEMENT EXECUTION, CONTROL WILL PASS TO THE STATEMENT FOLLOWING THE 'END CASE'. IN THE EVENT THAT THE VALUE OF <EXPRES> IS DETERMINABLE AT COMPILE TIME, AND CONTAINS NO PRESET ITEMS, THE SELECTION WILL BE MADE AT COMPILE TIME.

211 <CYCLE STAT> ::= <CYCLE HEAD><QUALIFIER><XEQ STAT LST> NEXT
<GROUP NAME> :

IF THE <GROUP NAME> FOLLOWING 'NEXT' DOES NOT MATCH THAT WITHIN <CYCLE HEAD> THEN A WARNING MESSAGE RESULTS. IF NO ERRORS HAVE OCCURRED, THE <GROUP NAME> FOLLOWING 'NEXT' IS CHANGED ON OUTPUT TO THAT OF THE <GROUP HEAD>. WHEN CONTROL PASSES OUT OF THE <XEQ STAT LST> IT RETURNS TO THE TESTING AND INCREMENTATION (IF ANY) OF THE <QUALIFIER>.

216 <CONTROL PART> ::= WHILE <CONDITION>
217 ! <FOR PART> <RANGE LIST>
218 ! <EMPTY>

IN CASE 217, IF THE RANGE LIST IS NOT EXHAUSTED, THE NEXT VALUE OF THE RANGE LIST IS ASSIGNED TO <IDENTIFIER>. IF THE RANGE LIST IS EXHAUSTED, OR IF CASE 216 AND THE <CONDITION> IS NOT TRUE, THEN CONTROL PASSES TO THE FIRST STATEMENT AFTER THE 'NEXT' <GROUP NAME> : . OTHERWISE CONTROL PASSES TO THE FIRST STATEMENT OF THE <XEQ STAT LST>.

219 <FOR PART> ::= FOR <IDENTIFIER> : =

THE <IDENTIFIER> MUST BE DECLARED TO BE OF TYPE INT OR INDEX.

```
220     <EXPRES> ::= <TERM>
221                ! - <TERM>
222                ! <EXPRES> + <TERM>
223                ! <EXPRES> - <TERM>
```

ALL TERMS OF AN EXPRESSION MUST HAVE COMPATIBLE TYPE. IF THAT TYPE IS INT, SINT, OR INDEX, THE TYPE OF THE EXPRESSION IS DETERMINED BY THE TYPE OF THE ITEM TO WHICH THE RESULT IS TO BE ASSIGNED, OR IF NO ASSIGNMENT IS TO BE MADE, THE TYPE OF THE EXPRESSION IS INT.
RULE 221 IS THE UNARY MINUS AND IS VALID ONLY FOR NUMERIC TERMS. AND POWERSET TERMS, MEANING NEGATION AND COMPLEMENTATION RESPECTIVELY.
FOR NUMERIC TYPES THE MEANING OF THE OPERATORS '+' AND '-' IS THE STANDARD MEANING. FOR STRING TYPES, '+' AND '-' REPRESENT STRING CONCATENATION AND EXCISION RESPECTIVELY. THE MEANING OF THE LATTER IS THAT THE RESULT OF A-B, WHERE A AND B ARE STRINGS, CONSISTS OF THOSE CHARACTERS OF A WHICH DO NOT FORM THE LEFT-MOST OCCURRENCE IN A OF THE STRING B.
FOR POWERSET TYPES, THE OPRATIONS '+' AND '-' CORRESPOND TO SET UNION AND RELATIVE COMPLEMENT RESPECTIVELY.
FOR ALL OTHER TYPES 221, 222, AND 223 ARE NOT ALLOWED.

```
224     <TERM> ::= <PRIMARY>
225              ! <TERM> * <PRIMARY>
226              ! <TERM> / <PRIMARY>
227              ! <TERM> \ <PRIMARY>
```

ALL PRIMARIES IN A TERM MUST BE OF COMPATIBLE TYPE. FOR NUMERIC PRIMARIES THE OPERATORS '*','/', AND '\' REPRESENT MULTIPLICATION, DIVISION, AND REMAINDER RESPECTIVELY. FOR STRING PRIMARIES, ONLY '/' AND '\' ARE ALLOWED AS OPERATORS, REPRESENTING HEADER AND TRAILER OPERATIONS. A/B CONSISTS OF THOSE CHARACTERS OF A WHICH PRECEDE THE LEFT-MOST OCCURRENCE OF THE CHARACTERS OF B WITHIN THE STRING A. A\B CONSISTS OF THOSE CHARACTERS OF A WHICH FOLLOW THE LEFT-MOST OCCURRENCE OF THE CHARACTERS OF B WITHIN THE STRING A.
FOR POWERSET PRIMARIES ONLY THE OPERATOR '*' IS ALLOWED, MEANING SET INTERSECTION.

```
228     <PRIMARY> ::= <STORE REF>
229                 ! RANGE
```

'RANGE' IS ALLOWED AS A PRIMARY ONLY WITHIN THE <EXPRES> OF 7/.

230 ! <NUMBER>

<NUMBER> IS ALLOWED AS A PRIMARY ONLY WITHIN THE <EXPRES> OF 32 AND 76:77.

231 ! <PARAM IDENTIFIER>

<PARAM IDENTIFIER> IS ALLOWED AS A PRIMARY ONLY IF A VALUE HAS BEEN ASSIGNED TO IT IN A DECLARATION PRECEDING THE APPEARANCE OF THIS EXPRESSION.

232 ! <QSTRING>

<QSTRING> IS ALLOWED AS A PRIMARY ONLY WITHIN THE <EXPRES> OF 32 AND 76:77.

233 ! MIN (<EXPRES> , <EXPRES>)
234 ! MAX (<EXPRES> , <EXPRES>)

THE TWO EXPRESSIONS MUST BE OF NUMERIC TYPE, AND THE RESULT IS OF THE SAME TYPE, BEING THE SMALLER AND LARGER OF THE TWO VALUES OF THE EXPRESSIONS RESPECTIVELY.

235 ! ASCII6(<EXPRES>)

THIS IS A BUILT-IN CHARMAP. THE EXPRESSION MUST EVALUATE TO A ONE CHARACTER STRING. THE RESULT IS THE INTEGER INTO WHICH THAT CHARACTER MAPS.

236 ! CONVIS (<EXPRES>)
237 ! CONVSI (<EXPRES>)

THESE ARE INTEGER-TO-STRING AND STRING-TO-INTEGER CONVERSIONS. THE ARGUMENT OF CONVIS MUST EVALUATE TO A NON-NEGATIVE INTEGER. IF THE ARGUMENT IS FOUND TO BE NEGATIVE, AN ERROR RESULTS AT COMPILE TIME OR UNDEFINED RESULTS AT RUN TIME.
THE RESULT IS A STRING OF LENGTH THE REQUIRED NUMBER OF DIGITS. THE ARGUMENT OF CONVSI MUST EVALUATE TO A STRING. THE INTEGER VALUE DEFINED BY THAT STRING IS OBTAINED BY SCANNING THE STRING FROM THE LEFT. THE FIRST NON-BLANK CHARACTER INTITIATES THE DIGIT STRING, AND FROM THAT POINT THE FIRST NON-DIGIT (OR THE END OF THE STRING) TERMINATES THE DIGIT STRING. NOTE THAT THE DIGIT STRING IS EMPTY IF THE FIRST NON-BLANK IS NOT A DIGIT OR IF THE STRING CONTAINS ALL BLANKS. THE VALUE ASSIGNED TO AN EMPTY DIGIT STRING IS ZERO. IF THE RESULTING INTEGER IS LARGER THAN TEN DIGITS AN ERROR IS GENERATED AT COMPIULE TIME, OR UNDEFINED RESULTS AT RUN TIME.

238 * ! FLAGVAL (<IDENTIFIER>)

THIS IS NOT SUPPORTED IN THE CURRENT IMPLEMENTATION.

239 ! ALL

THIS IS ALLOWED ONLY FOR POWERSET TERMS, AND REPRESENTS THE ALL-INCLUSIVE SET.

220 ! ABS(<EXPRES>)

THE ARGUMENT MUST EVALUATE TO A NUMERIC TYPE ITEM. THE VALUE OF THE PRIMARY IS THE ABSOLUTE VALUE OF THE EXPRESSION.

221 ! EMPTY

THIS IS ALLOWED ONLY FOR POWERSET TERMS, AND REPRESENTS THE EMPTY SET.

222 ! (<EXPRES>)

THIS <EXPRES> MUST SATISFY THE SAME RESTRICTIONS ON <PRIMARY> AS THE <EXPRES> WHICH CONTAINS IT.

5.2.3.4 PROCESSING - PROGRAMMER OUTPUTS

FOR EACH BLOCK COMPILED, ONE OUTPUT, THE REFORMATTED SOURCE, IS ALWAYS PRODUCED. THIS IS A LISTABLE FILE WHICH ALSO CONTAINS ALL ERROR AND WARNING MESSAGES GENERATED BY THE COMPILER. IN NORMAL OPERATION THIS IS RETURNED TO THE SYSTEM TEN AS A PUNCH FILE AND IS ROUTED TO DISC BY THE STANDARD RJF DISC PROCEDURES. WHILE DEBUGGING, IT IS MOST CONVENIENT TO EDIT THIS OUTPUT FILE WITH ANY NECESSARY CORRECTIONS AND THEN TO RESUBMIT THIS AS INPUT TO THE COMPILER. DURING COMPILATION, THE DISCOVERY OF AN ERROR IN THE PROGRAM WILL CAUSE THE REFORMATTING TO STOP. OUTPUT THEN PROCEEDS SIMPLY AS A COPY OF THE INPUT FROM COLUMN 1 THROUGH COLUMN 74 OF EACH LINE, WITH A NEWLY CALCULATED LINE NUMBER IN COLUMNS 75-78.

PRIOR TO EDITING THIS FILE, IT IS NECESSARY TO CONVERT IT TO AN S-TYPE FILE. THE FIRST EDIT STEP SHOULD BE THE COMMAND

    USE SINGLE.RJRSLT.

THIS RECONVERTS '&' AND '%' TO '\' AND '↑' RESPECTIVELY.

FOUR TYPES OF MESSAGES MAY BE INCLUDED IN THE SOURCE OUTPUT LISTING:

- INFORMATION
- WARNING
- ERROR
- FATAL ERROR

THE LAST CATEGORY, FATAL ERROR, MEANS THAT COMPILATION OF THIS BLOCK IS ABORTED. NO ATTEMPT WILL BE MADE TO FIND FURTHER ERRORS IN THIS BLOCK.

THE REFORMATTED SOURCE FILE WILL BEGIN WITH TWO LINES CONTAINING COMPILER HEADING WHICH MAY OR MAY NOT BE DELETED BEFORE RESUBMISSION, SINCE THEY ARE IN THE FORM OF COMMENTS. ANY ERROR MESSAGES BEGIN WITH '***' IN COLUMNS 1 THROUGH 3. THESE MESSAGES MUST BE DELETED BEFORE RESUBMISSION. AT THE POINT WHERE THE FIRST ERROR WAS DETECTED, SOME DUPLICATION OF CODE MAY EXIST. THE REFORMATTER, ON DETECTING THE FIRST OCCURRENCE OF ERROR, OUTPUTS ALL REMAINING FORMATTED INFORMATION UP TO THE LAST TERMINAL CONSTRUCT SCANNED PRIOR TO THE ERROR DETECTION. THE REFORMATTER THEN REVERTS TO SIMPLY LISTING THE INPUT LINES, AND THIS MAY PRODUCE REPETITION OF A FEW CONSTRUCTS. THE ERROR MESSAGE ITSELF WILL PRINT BEFORE THE PRINT OF THE LINE ON WHICH THE ERROR OCCURRED.

WARNING MESSAGES BEGIN WITH '0***WARNING', AND INFORM THE PROGRAMMER OF CONDITIONS WHICH MAY BE ERRONEOUS, ALTHOUGH THE COMPILER MAY TAKE ACTION TO PREVENT THE RECURRENCE OF THE CONDITION. THIS HAPPENS IN THE CASE OF NON-MATCHING GROUP NAMES ON 'BEGIN...END' OR 'CYCLE.....NEXT' CONSTRUCTS, WHERE THE REFORMATTER SIMPLY CHANGES THE NAME ON THE GROUP TERMINATOR TO MATCH THAT OF THE GROUP INITIATOR. THE WARNING MESSAGE NEED NOT BE DELETED PRIOR TO RESUBMISSION. BECAUSE OF ITS FORM IT WILL BE IGNORED AND THROWN AWAY BY THE SCANNER.

AT THE END OF EACH BLOCK TWO ADDITIONAL LINES OF COMPILER INFORMATION ARE PRINTED. THESE AGAIN ARE IN THE FORM OF COMMENTS AND NEED NOT BE DELETED PRIOR TO RESUBMISSION.

IN ADDITION TO THE SOURCE OUTPUT, FOUR OPTIONAL OUTPUTS MAY BE OBTAINED, CONTROLLED BY THE SAVE, CODE, ALIST, AND CHECK COMPILER OPTIONS AND THE CHECK OPTION ON DECLARATIONS: THE SAVE OPTION PUTS A COMPILED FORM OF THE SYMBOL TABLE

ON A SYSTEM FILE AT THE 370. ONCE THIS HAS BEEN DONE, IT IS NOT NECESSARY TO RESUBMIT THAT BLOCK WHEN COMPILING A SUBORDINATE BLOCK.

THE CODE OPTION PRODUCES A PUNCH FILE OF RELOCATABLE CODE AND LOADABLE DATA WHICH CONSTITUTE THE OBJECT FORM OF THE SOURCE BLOCK. THIS FILE MAY THEN BE INPUT TO THE LINK EDITOR WHEN A LOAD MODULE FOR A COMPLETE PROGRAM IS TO BE PREPARED.

THE ALIST OPTION PRODUCES AN ASSEMBLY-LIKE LISTING OF THE DATA DECLARATIONS AND ALLOCATIONS FOR THAT BLOCK AND ALL BLOCKS TO WHICH IT IS SUBORDINATE. IN ADDITION, IF THE CODE OPTION IS ON, THE ALIST OPTION PRODUCES A LISTING OF THE GENERATED CODE IN ASSEMBLY-LIKE FORM. FOR AN EXAMPLE OF THE ALIST FORMAT, SEE APPENDIX A, 'ASSEMBLY LANGUAGE CONSIDERATIONS'.

THE CHECK OPTIONS ON DECLARATIONS CAUSE A CONTROLLED CROSS-REFERENCE LISTING FOR EACH BLOCK, SHOWING, FOR EACH ITEM IN ITS DATA SPACE WHICH HAS A CHECK SPECIFIED, THE LINE NUMBER OF EACH ACTION WHICH CAUSES A REFERENCE OF THE KIND SPECIFIED TO THAT DATA ITEM. THE CHECK COMPILER OPTION PRODUCES A SIMILAR LISTING FOR ALL PARAMETERS AND DATA ITEMS.

THE RESULTS OF A COMPILATION CONSISTS OF TWO OR THREE FILES, NO MATTER HOW MANY BLOCKS WERE INCLUDED IN THE RUN. THE FIRST FILE IS THE HASP HEADER RECORD, WHICH IS USED ONLY TO IDENTIFY THE JOB. THE NEXT FILE CONTAINS THE DISPLAYABLE OUTPUT: REFORMATTED SOURCE, COMPILER MESSAGES, AND A-LIST AND CHECK LISTINGS. IF THE CODE OPTION IS USED, ANOTHER FILE WILL CONTAIN THE RELOCATABLE CODE FOR THOSE BLOCKS.

5.2.4 TERMINATIONS

5.2.4.1 TERMINATIONS - OPERATOR INPUT

NONE.

5.2.4.2 TERMINATIONS - OPERATOR OUTPUT

STANDARD OS/VS2 OPERATOR OUTPUT FOR JOB TERMINATION.

5.2.4.3 TERMINATIONS - PROGRAMMER INPUTS

THE PROGRAMMER MUST SUPPLY THE STANDARD OS/VS2 JOB FILE TERMINATOR, WHICH IS A RECORD CONSISTING OF '/*' IN COLS. 1-2, AND BLANKS IN THE REMAINING COLS. THIS REPLACES THE '*!*' ENDING THE LAST COMPILATION BLOCK. ON OUTPUT THE '/*' IS ALWAYS REPLACED BY '*!*'.

5.2.4.4 TERMINATIONS - PROGRAMMER OUTPUTS

THE ONLY PROGRAMMER OUTPUT IS THE JOB TERMINATION BANNER PAGE ON THE REMOTE OR LOCAL PRINTER.

5.2.5 INTERNAL STRUCTURAL FEATURES

THIS PRODUCT IS BASED ON THE XCOM COMPILER STRUCTURE. THE ALGORITHMS HAVE BEEN MODIFIED ONLY TO THE EXTENT NEEDED TO SUPPORT THE SPECIFIC FEATURES OF THE SINGLE LANGUAGE, TO PROVIDE FOR SOURCE TEXT REFORMATTING, AND TO GENERATE CODE FOR THE SYSTEM TEN.

5.2.6 PROVISIONS

5.2.6.1 RELIABILITY

NO SPECIAL PROVISIONS HAVE BEEN MADE FOR PROTECTION AGAINST ENVIRONMENTAL FAILURES, BEYOND THOSE EXIXTING IN OS, HASP, AND RJF DISC. ALL ERRORS IN SOURCE INPUT ARE DETECTED AND REPORTED. NO INPUT ERROR CAN CAUSE THE COMPILER TO FAIL OR TO CRASH THE SYSTEM. NOTE, HOWEVER, THAT ATTEMPTING TO TRANSMIT A FILE VIA RJF DISC WHICH CONTAINS '\' OR '↑'

CHARACTERS WILL CAUSE TRANSMISSION FAILURE.

5.2.6.2 RESTARTABILITY

NO RESTART CAPABILITY IS PROVIDED.

5.2.6.3 MAINTAINABILITY

THE SINGLE CROSS-COMPILER IS WRITTEN IN PL/I AND BASED ON THE VERY MODULAR STRUCTURE OF XCOM. PL/I LEVEL H COMPILER PROVIDES A NUMBER OF VALUABLE DEBUGGING AIDS, AND THE COMPILER CONTAINS BUILT-IN TRACING CAPABILITY CONTROLLED BY THE '?' FEATURE MENTIONED IN 3.2.2.1.
SINCE THIS PRODUCT IS NOT PLANNED FOR INSTALLATION ON USER EQUIPMENT, NO PROVISIONS FOR MAINTAINING PRODUCT I.D. HAVE BEEN MADE.

6. GENERATED SOFTWARE

6.1 SYSTEM CONSIDERATIONS

SINCE THE GENERATED SOFTWARE IS GENERAL SYSTEMS PROGRAMMING TYPE CODE, VERY FEW RESTRICTIONS OR CONVENTIONS ARE REQUIRED.

6.1.1 HARDWARE CONSIDERATIONS

THE GENERATED CODE IS TO BE LINK EDITED AND LOADED. HARDWARE TO SUPPORT THESE OPERATIONS, PLUS A PARTITION TO RUN IN, IS ALWAYS REQUIRED. ADDITIONAL HARDWARE REQUIREMENTS MAY BE IMPOSED BY A PARTICULAR PROGRAM.

6.1.2 SOFTWARE CONSIDERATIONS

THE CODE PRESUMES A DMF II ENVIRONMENT, INCLUDING COMMON LIOCS (THE LATEST CURRENT VERSION), AND THE LINK EDITOR. DETAILS ON LINKAGE OF SINGLE BLOCKS FROM OR TO ASSEMBLY-CODED ROUTINES ARE GIVEN IN APPENDIX A, 'ASSEMBLY LANGUAGE CONSIDERATIONS'.

6.2 EXTERNAL ORGANIZATION

EXTERNAL ORGANIZATION IS ARBITRARY WITHIN THE SYSTEM OF BLOCK RELATIONSHIPS DESCRIBED IN 3.2.3.

6.2.1 GENERATION

THE ONLY CONTROLS ON GENERATION ARE THOSE OF LINK EDITOR.

6.2.1.1 CONDITIONAL ASSEMBLY OPTION

IF IT IS PLANNED TO INCORPORATE ASSEMBLY CODED MODULES INTO A SINGLE PROGRAM, THEY MAY HAVE SUCH OPTIONS. CODE PRODUCED BY THE SINGLE COMPILER HAS NONE, BEING IN RELOCATABLE FORM.

6.2.1.2 CUSTOMIZATION

NO CUSTOMIZATION FEATURES FOR OBJECT CODE ARE PROVIDED.

6.2.1.3 INSTALLATION

INSTALLATION CONSISTS OF LOADING THE CODE IN NORMAL DMF II FASHION VIA THE CONVERSATIONAL LOADER.

6.2.2 INVOCATION

NO SPECIAL PROVISIONS ARE MADE BEYOND THOSE OF THE DMF II SYSTEM.

6.2.3 PROCESSING

NO SPECIAL PROVISIONS ARE MADE BEYOND THOSE OF THE DMF II SYSTEM.

6.2.4 TERMINATION

NO SPECIAL PROVISIONS ARE MADE BEYOND THOSE OF THE DMF II SYSTEM.

6.2.5 INTERNAL STRUCTURAL FEATURES

NO GENERAL FEATURES EXIST UNDER THE CURRENT IMPLEMENTATION.

6.2.6 PROVISIONS

NO SPECIAL PROVISIONS ARE MADE FOR RELIABILITY, RESTARTABILITY,

OR MAINTAINABILITY OF GENERATED CODE.

SINGLE CROSS-COMPILER ERS

APPENDIX A

ASSEMBLY LANGUAGE CONSIDERATIONS

1. EXAMPLES OF ALIST AND CODE OUTPUT

ALL OF THIS APPENDIX WILL BE BASED ON THE FOLLOWING EXAMPLE OF SINGLE CODE. THE CODE ITSELF DOES NO RECOGNIZABLY USEFUL WORK, BUT IS INTENDED ONLY TO POINT UP THE VARIOUS CONSIDERATIONS IN RELATING SINGLE CODE TO MACHINE ADDRESSES AND IN LINKING SINGLE PROCEDURES FROM AND TO ASSEMBLY CODED ROUTINES.

SOURCE INPUT.

```
      OPTIONS(SAVE)
0    CONTEXT NULCON
1     TYPE
2      EXAMP1 = TASK LINK
2      END EXAMP1
1     END TYPE
.!.
```

```
      OPTIONS(ALIST)
0    TASK EXAMP1
1     PARENT NULCON
1     PARAMETER
2      £ASIZE          :9
2      £BSIZE          :15
2      £MAXAB          :MAX(£ASIZE,£BSIZE)
1     END PARAMETER
1     TYPE
2      A.REC = RECORD
3       HEADER         :STRING[£BSIZE]
3       BODY           :ARRAY[£BSIZE] OF INT[£ASIZE]
2      END A.REC
2      EXAMP2 = PROCEDURE LINK
3       N              :INT[£BSIZE]
2      ASSIGNS
3       S              :STRING[£ASIZE]
2      END EXAMP2
1     END TYPE
1     PRESET
2      INITN          :INT[£BSIZE]              :£BSIZE=MIN(£ASIZE,
          £BSIZE)
2      INITS          :STRING[£ASIZE]           :'LASTVAL'
```

```
* 1     END PRESET
* 1     VARIABLE
* 2      PARM              :EXAMP2
* 1     END VARIABLE
* 1     PARM.N:=INITN
* 1     PARM.S:=INITS
* 1<    CALL EXAMP2(PARM)
* 1=    STOP              ! CURRENTLY LACK CODE GENERATION FOR 'STOP'
* +!+
*
* 0    PROCEDURE EXAMP2
* 1     PARENT EXAMP1
* 1     CONSTANT
* 2      C1                :INT[£ASIZE]
* 2      CZ                :INT[£BSIZE]
* 1     END CONSTANT
* 1     VARIABLE
* 2      X,Y               :INT[£MAXAB]
* 2      Z                 :INT[£ASIZE]
* 2      A+RAY             :ARRAY[INITN] OF AREC
* 2      W                 :INDEX=>A+RAY
* 1     END VARIABLE
* 1     X:=@.N=C1
*
*
* 1     @.S(CZ):=@.S(X)
* 1B0   RETURN
* +!+
*
* SOURCE OUTPUT.
*
*     THE RESULT OF COMPILING THE ABOVE CODE IS AS FOLLOWS (NOTE THAT THE
*     CODE OPTION IS ON BY DEFAULT):
*
*        OPTIONS(SAVE)
* 0    CONTEXT NULCON
* 1     TYPE
* 2      EXAMP1 = TASK LINK
* 2      END EXAMP1
* 1     END TYPE
* +!+
*        OPTIONS(ALIST)
* 0    TASK EXAMP1
* 1     PARENT NULCON
* 1     PARAMETER
* 2      £ASIZE            :9
*
*
*
```

APPROVED

```
* 2      £BSIZE          :15                                              *
* 2      £MAXAB          :MAX(£ASIZE,£BSIZE)                              *
* 1    END PARAMETER                                                      *
* 1    TYPE                                                               *
* 2      A-REC = RECORD                                                   *
* 3       HEADER         :STRING[£BSIZE]                                  *
* 3       BODY           :ARRAY[£BSIZE] OF INT[£ASIZE]                    *
* 2      END A-REC                                                        *
* 2      EXAMP2 = PROCEDURE LINK                                          *
* 3       N              :INT[£BSIZE]                                     *
* 2      ASSIGNS                                                          *
* 3       S              :STRING[£ASIZE]                                  *
* 2      END EXAMP2                                                       *
* 1    END TYPE                                                           *
* 1    PRESET                                                             *
* 2      INITN          :INT[£BSIZE]              £BSIZE=MIN(£ASIZE,     *
*           £BSIZE)                                                       *
* 2      INITS          :STRING[£ASIZE]           :'LASTVAL'             *
* 1    END PRESET                                                         *
* 1    VARIABLE                                                           *
* 2      PARM           :EXAMP2                                          *
* ********** DECLARATIVES FOR TASK EXAMP **********                      *
* *                                                                       *
* * OFFSETS FOR RECORD EXAMP1 FOLLOW                                     *
* *                                                                       *
*          EXAMP1 EXTERNAL                                               *
*          £ASIZE EQU    9                                               *
*          £BSIZE EQU    6                                               *
*          £MAXAB EQU                                                    *
* * OFFSETS FOR RECORD A-REC  FOLLOW                                     *
*          HEADER EQU    0                                               *
*          BODY   EQU   15                                               *
* *                                                                       *
* * OFFSETS FOR RECORD EXAMP2 FOLLOW                                     *
*          N      EQU    0                                               *
*          S      EQU    2                                               *
* *                                                                       *
*          EXAMP2 EXTERNAL                                               *
*    1     INITN  DM      N2'06'                                         *
*    3     INITS  DM      C9'LASTVAL  '                                  *
*   12     PARM   DM      N11                                            *
*        END                                                            *
* 1    END VARIABLE                                                       *
*         EXAMP1 ENTRY                                                  *
*    0    EXAMP1 BC      *+10(5,00),*+10(5,00)                          *
*   10           MC      INITN(02,00),PARM+0(,00)                       *
* 1    PARM.N:=INITN                                                      *
*                                                                       *
*                                                                       *
*                                                                       *
```

```
   20              MC     INITS(09,00),PARM+2(,00)
 1     PARM.S:=INITS
   30              MC     ←PARM(04,00),←REG3(,00)
   40              BC     EXAMP2+1(6,00),EXAMP2+10(5,00)
 1<    CALL EXAMP2(PARM)
 1=    STOP             * CURRENTLY LACK CODE GENERATION FOR 'STOP'
 ←!←
 0    PROCEDURE EXAMP2
 1     PARENT EXAMP1
 1     CONSTANT
 2      C1              :INT[£ASIZE]                  :1
 2      CZ              :INT[£BSIZE]                  :0
 1     END CONSTANT
 1     VARIABLE
 2      X,Y             :INT[£MAXAB]
 2      Z               :INT[£ASIZE]
 2      A←RAY           :ARRAY[INITN] OF A←REC
 2      W               :INDEX=>A←RAY
 ********** DECLARATIVES FOR PROCEDURE EXAMP2 **********
 *
 * OFFSETS FOR RECORD EXAMP1 FOLLOW
 *
*          EXAMP1 EXTERNAL
*          £ASIZE EQU            9
*          £BSIZE EQU           15
*          £MAXAB EQU            1
 * OFFSETS FOR RECORD A←REC  FOLLOW
*          HEADER EQU   0
*          BODY   EQU  15
 *
 * OFFSETS FOR RECORD EXAMP2 FOLLOW
*          N      EQU   0
*          S      EQU   2
 *
*          EXAMP2 EXTERNAL
*    1     INITN  DM      N2'06'
*    3     INITS  DM      C9'LASTVAL  '
*   12     PARM   DM      N11
*   23     C1     DM      N1'1'
*   24     CZ     DM      N2'00'
*   26     X      DM      N2
*   28     Y      DM      N2
*   30     Z      DM      N1
*   31     A←RAY  DM     6A30
*  211     W      DM      A4
*       END
 1.    END VARIABLE
```

```
*          EXAMP2 ENTRY
*     0    EXAMP2 BC     *+10(5,00),*+10(5,00)
*    10           MC     0+0(02,03),X(,00)
*    20           S      C1(1,00),X(2,00)
* 1      X:=@.N-C1
*    30           MC     +X(04,00),+REG1(,00)
*    40           A      +REG3(4,00),+REG1(4,00)
*    50           MC     0+2(01,01),0+2(,03)
*
*
*
* 1      @.S(CZ):=@.S(X)
*    60           BC     EXAMP2(5,00),EXAMP2(5,00)
* 180  RETURN
* +!+
*
*
*
*
*
* 2.  ADDRESS CALCULATIONS
*
*
* THE ALIST FOR THE DECLARATIVES OF A BLOCK SHOWS ABSOLUTE
* ADDRESSES OF ALL THE DATA ACCESSIBLE TO THAT BLOCK, I.E., DECLARED
* WITHIN THAT OR SOME HIGHER LEVEL BLOCK.  COMMON ADDRESSES ARE
* SUFFIXED BY 'C'.  IN ORDER TO PRESERVE THE CONVENTIONS OF LOW
* CORE USAGE IN PARTITION AND COMMON MEMORY, DUMMY ARRAYS MUST BE
* DECLARED IN HIGH LEVEL BLOCKS, E.G. AN ARRAY WHOSE LENGTH COVERS ALL
* THE LOW CORE AREA SPECIALLY RESERVED.
* CODE ADDRESSES ARE INDICATED IN RELOCATABLE FORM,
* BEFORE EACH GENERATED INSTRUCTION FOR THAT BLOCK.  THE SOURCE LINE
* CORRESPONDING TO A GROUP OF INSTRUCTIONS FOLLOWS THE INSTRUCTIONS.
* IT IS IMPORTANT TO REALIZE THAT THE COMPILED FORM OF A BLOCK CONSISTS
* OF TWO LOAD DECKS.  ONE IS THE (RELOCATABLE) DECK OF LOADABLE DATA
* DECLARED WITHIN THAT BLOCK.  ALTHOUGH IN RELOCATABLE FORM, THIS DECK
* MUST BE LOADED AT THE PROPER ADDRESS, SINCE REFERENCES TO THE DATA
* WITHIN THE SECOND DECK, I.E., THE RELOCATABLE CODE DECK, ARE ABSOLUTE.
* THE CODE DECK IS ITSELF FULLY RELOCATABLE, AND CONTAINS ONE ENTRY
* POINT, WHOSE NAME IS THE BLOCK NAME.
*
*
*
*
* 3.  LINKING TO ASSEMBLY-CODED BLOCKS
*
*
* AN ASSEMBLY CODED-BLOCK COULD REPLACE THE 'EXAMP2' PROCEDURE, FOR
*
*
*
```

EXAMPLE. SUCH A BLOCK WOULD DECLARE 'EXAMP2' AS ITS ENTRY POINT. ITS FIRST INSTRUCTION WOULD BE THE BRANCH WHICH RETURNS TO ITS CALLER, AND THE PROCEDURE ITSELF WOULD BEGIN AT LOCATION EXAMP2+10. THE ADDRESS OF ITS PARAMETER RECORD IS CONTAINED IN INDEX REGISTER 3. OFFSETS FOR THE FIELDS OF THAT RECORD MAY BE OBTAINED FROM THE ALIST FOR ITS PARENT. DATA ADDRESSES FOR OTHER DATA ITEMS THE PROCEDURE MAY REFERENCE MAY ALSO BE OBTAINED FROM THE PARENT'S ALIST. RETURN FROM THE BLOCK IS EFFECTED BY EXECUTING A BRANCH TO EXAMP2.

4. LINKING FROM ASSEMBLY-CODED PROGRAMS

WRITING A PROCEDURE IN SINGLE WHICH IS CALLED FROM AN ASSEMBLY-CODED PROGRAM REQUIRES THAT THE CALLING PROGRAM OBSERVE THE CALLING SEQUENCE EXPECTED BY SINGLE PROCEDURES. THE LINK IS MADE TO THE ENTRY POINT AND CONTROL PASSES TO ENTRY POINT+10. THE ADDRESS OF THE ARGUMENT RECORD IS PLACED IN INDEX REGISTER 3 BEFORE DOING THE LINK. IN ADDITION, THE PROCEDURE MUST HAVE A PARENT, CODED IN SINGLE, WHICH INCLUDES THE LINK DECLARATION FOR THAT PROCEDURE SO THAT OFFSETS FOR THE FIELDS OF THE PARAMETER RECORD ARE AVAILABLE TO THE COMPILER. (OF COURSE, A PARENT IS NEEDED EVEN IF THE PROCEDURE HAS NO ARGUMENTS SIMPLY TO SATISFY THE SYNTAX). THIS PARENT WILL TYPICALLY BE A CONTEXT BLOCK. THE CONTEXT MAY ALSO SPECIFY DATA ITEMS WHICH ARE NOT ARGUMENTS TO THE PROCEDURE BUT WHICH IT MUST ACCESS. IN ORDER THAT THE PROPER DATA ADDRESSES BE AVAILABLE TO THE COMPILER, DUMMY DECLARATIONS MUST BE MADE TO INSURE THAT THE DATA NAMES ARE ALLOCATED TO THE CORRECT ADDRESSES IN ABSOLUTE DATA SPACE.

SINGLE ERS

APPENDIX B

DEFINITION CROSS REFERENCE TABLE

APPENDIX B

A

V#	CONSTRUCT	DEFINED IN	USED IN DEFINITION
39	(	TERMINAL	102
67	.	TERMINAL	173
74	<	TERMINAL	197 204
2	(	TERMINAL	3 56 159 181 196 233:238 240 242
82	+	TERMINAL	221
85	\	TERMINAL	227
83	*	TERMINAL	225
3	)	TERMINAL	3 55 159 178:179 [illegible] 233:238 240 242
36	-	TERMINAL	[illegible] 22 224
84	/	TERMINAL	[illegible]
4	,	TERMINAL	[illegible] 35 75 79 107 111 150 157:158 171 182 233:234
75	!	TERMINAL	205
96	←!←	TERMINAL	1:2
37	>	TERMINAL	99 198 203
20	:	TERMINAL	33 67 77 118:119 128:129 166 206:207 211 213 219
21	£	TERMINAL	36
68	@	TERMINAL	177
7	=	TERMINAL	9 43 166 199 203:205 219
94	ABS	TERMINAL	240
93	ALL	TERMINAL	239
72	AND	TERMINAL	192
38	ARRAY	TERMINAL	100
159	<ARRAY TYPE>	100	92
89	ASCII64	TERMINAL	235
133	<ASSIGNS>	54	49 51
26	ASSIGNS	TERMINAL	54
171	<BAL STAT>	122:125	120 131 137:138 183 185
76	<BLOCK HEAD>	207	206
174	<BLOCK STAT>	206	123
193	<BY>	169	167
66	BY	TERMINAL	169
192	<BY PART>	167:168	164:165
52	CALL	TERMINAL	140

77	CASE	TERMINAL	208:209
210	<CASE>	209	208
211	<CASE OF>	210	208
175	<CASE STAT>	208	124
13	CHARMAP	TERMINAL	21 57
111	<CHARMAP DEF>	57	21
41	CHECK	TERMINAL	104
146	<CHECK SPEC>	104:105	73 90
99	<COMPILATION>	10:11	1:2
139	<CON DECL LST>	64:65	62 65 68
140	<CON DESCRIP>	66	64:65
203	<COND FACT>	193:194	191:192
204	<COND PRIM>	195:196	193:194
202	<COND TERM>	191:192	189:190 192
190	<CONDITION>	189:190	161 186 190 196 216
114	<CONST DECL>	62	24
16	CONSTANT	TERMINAL	24 63
138	<CONSTANT>	63	62
8	CONTEXT	TERMINAL	12
105	<CONTEXT BLK>	17:18	10:11
104	<CONTEXT NAME>	12	10
214	<CONTROL PART>	216:217	213
90	CONVIS	TERMINAL	236
91	CONVSI	TERMINAL	237
79	CYCLE	TERMINAL	212
212	<CYCLE HEAD>	211	211
176	<CYCLE STAT>	211	125
109	<DECL LST>	19:20	17 20
65	DECR	TERMINAL	165
110	<DEF ITEM>	21:28	19:20
46	DEFER	TERMINAL	132
186	<DISABLE>	152	145
61	DISABLE	TERMINAL	152
51	DO	TERMINAL	139
71	ELSE	TERMINAL	188
200	<ELSE>	188	183 185
95	EMPTY	TERMINAL	241
97	<EMPTY>	TERMINAL	16 18 53 88 105
			160 168 180 215
			218
185	<ENABLE>	151	144
60	ENABLE	TERMINAL	151
12	END	TERMINAL	21:28 40:41 206 208
182	<EVENT>	161:162	133
123	<EXPRES>	220:222	32 76:77 82:84 87

			101 134 144:145
			156:157 163 167 175
			178:179 233:237 240
			242
195	<EXPRES PAIR>	178:180	172:173
197	<EXPRES PAIR (>	181	178:179
198	<EXPRES PAIR C>	182	178
161	<FAKE ARROW>	99	97
63	FINISH	TERMINAL	162
144	<FLAG>	72	70
19	FLAG	TERMINAL	27 72
117	<FLAG DECL>	70:71	1 1
145	<FLAG DESCRIP>	73	70
92	FLAGVAL	TERMINAL	88
81	FOR	TERMINAL	9
215	<FOR PART>	219	217
206	<GR OR EQ>	203	200
179	<GROUP NAME>	214:215	128:129 206:207
			211:212
170	<GROUP NUMBER>	118:119	116:117
5	<IDENTIFIER>	TERMINAL	6:8 12:15 28 36
			40:41 43 52 93
			97:98 108 110:111
			113 130:140 214 219
			238
135	<IDENTIFIER LST>	110:111	55 73 90 111
70	IF	TERMINAL	187
201	<IF>	187	186
199	<IF PART>	186	183:185
178	<IF STAT>	183	127
64	INCR	TERMINAL	164
33	INDEX	TERMINAL	97
57	INP	TERMINAL	147
30	INT	TERMINAL	94
187	<IO PARAMS>	156:157	147:148
40	)	TERMINAL	103
169	<LAB STAT LST>	116:117	117 208
44	LEAVE	TERMINAL	129
163	<LEFT SS>	102	101 175
25	LIKE	TERMINAL	52
23	LINK	TERMINAL	45:46

```
*-------------------------------------------------------------------------------*
*                                                                               *
*                                                                               *
*                                                                               *
* 130   <LINK>                  45:48           41                              *
* 131   <LINK DECL>             49:53           41                              *
*  49    LOCK                   TERMINAL        135  137                        *
* 184   <LOR>                   154             137:138                         *
* 207   <LT OR EQ>              204             201                             *
*                                                                               *
*  88    MAX                    TERMINAL        234                             *
*  87    MIN                    TERMINAL        233                             *
*                                                                               *
*  78    NEXT                   TERMINAL        211                             *
*  73    NOT                    TERMINAL        194                             *
* 208   <NOT EQ>                205             202                             *
*   6   <NUMBER>                TERMINAL        8  118  230                     *
*                                                                               *
* 162   <OF>                    109             100                             *
*  35    OF                     TERMINAL        98  109  210                    *
* 102   <OPT ITEM>              6:8             4:5                             *
* 101   <OPT LST>               4:5             3  5                            *
*   1    OPTIONS                TERMINAL        3                               *
* 100   <OPTIONS>               3               2                               *
*  62    OR                     TERMINAL        153:154  190                    *
*  58    OUT                    TERMINAL        148                             *
*  24    OVERLAY                TERMINAL        47:48                           *
*                                                                               *
* 112   <PARAM DEF>             29:30           22  30                          *
* 120   <PARAM DESCRIP>         30              29:30                           *
* 121   <PARAM ID LST>          34:35           32  35                          *
* 124   <PARAM IDENTIFIER>      35              34:35  231                      *
*  14    PARAMETER              TERMINAL         22  31                         *
* 119   <PARAMETER>             31              29                              *
* 180   <PARAMS>                159:160         130:132  140  146               *
*  11    PARENT                 TERMINAL        15                              *
* 107   <PARENT DECL>           15:16           11                              *
* 189   <PARM C>                158             156:157                         *
*  55    PASS                   TERMINAL        143                             *
*  34    POWERSET               TERMINAL        98                              *
* 160   <PQUALIF>               101             94:96  100                      *
*  17    PRESET                 TERMINAL        25  69                          *
* 143   <PRESET>                69              68                              *
*                                                                               *
* 115   <PRESET DECL>           68              25                              *
* 217   <PRIMARY>               228:242         223:227                         *
*   9    PROCEDURE              TERMINAL        13  45  48                      *
* 106   <PROG NAME>             13:14           11                              *
*                                                                               *
*  86   <QSTRING>                TERMINAL        232                             *
* 213   <QUALIFIER>             213             211                             *
*                                                                               *
*                                                                               *
*                                                                               *
*-------------------------------------------------------------------------------*
*    S I N G E R   !   SINGLE ERS                 ! SHEET    !  K01              *
*  B U S I N E S S !                              ! REVISION !-------------------*
*  M A C H I N E S !   APPENDIX B                 !    A     ! NEXT   6 ! SHEET  5 *
*-------------------------------------------------------------------------------*
```

27	RANGE	TERMINAL	81 229
149	<RANGE LIST>	78:79	77 79 217
148	<RANGE PART>	81	77
150	<RANGE SPEC>	80	78:79
128	<RECORD>	44	40
22	RECORD	TERMINAL	44
43	RECYCLE	TERMINAL	128
205	<REL OP>	197:202	195
177	<REP STAT>	163:165	126
152	<REPLACE>	166	81 163
53	RETURN	TERMINAL	141
164	<RIGHT SS>	103	101 175
98	<RUN>	1:2	NOT USED IN RHS OF ANY PRODUCTION
132	<SCALAR>	55	42
134	<SCALAR HEAD>	56	55
48	SET	TERMINAL	134
103	<SET>	9	7:8
183	<SET TO>	155	134
158	<SIMPLE TYPE>	93:98	91 100
173	<SIMPLE XEQ>	126:150	122
31	SINT	TERMINAL	95
181	<SOR>	153	131
166	<SPEC ITEM>	108	106:107
165	<SPEC LST>	106:107	104 107
45	START	TERMINAL	130:131
59	STATUS	TERMINAL	149:150
29	STEP	TERMINAL	89
156	<STEP>	87	87
154	<STEP PART>	87:88	83:84
54	STOP	TERMINAL	142
191	<STORE LST>	170:171	163:165 171
188	<STORE REF>	172:173	149:150 156:157 159 170:171 173 228
32	STRING	TERMINAL	96
167	<SUB HEAD>	113	112
42	SUBROUTINE	TERMINAL	113
118	<SUBROUTINE DEF>	112	28
10	TASK	TERMINAL	14 46:47
122	<TCOLON>	33	32 90
216	<TERM>	223:227	220:222 225:227
69	THEN	TERMINAL	186
155	<TO>	86	84

```
 28    TO                      TERMINAL          85:86  155
 56    TRANSFER                TERMINAL          146
 15    TYPE                    TERMINAL          23  39
157   <TYPE>                   91:92             90

113   <TYPE DEF>               37:38             23  38
126   <TYPE DESCRIPT>          40:42             37:38
125   <TYPE HEAD>              39                37
127   <TYPE NAME>              43                40:42

172   <UNBAL IF>               184:185           121  185
 50    UNLOCK                  TERMINAL          136  138

147   <VALUE GEN>              76:77             74:75
142   <VALUE LST>              74:75             [illegible]:76
151   <VALUE PART>             82:84             80
196   <VAR>                    176:177           174:175
116   <VAR DECL>               58                [illegible]
129   <VAR DECL LST>           59:60             40  49:51  58  60
137   <VAR DESCRIP>            90                59:60  66
194   <VAR REF>                174:175           172:173
 18    VARIABLE                TERMINAL          26  61
136   <VARIBLE>                61                58
141   <VCOLON>                 67                66

 47    WAIT                    TERMINAL          133
 80    WHILE                   TERMINAL          216

168   <XEQ STAT>               120:121           14:117  184
108   <XEQ STAT LST>           [illegible]:115   11  112  115  206
                                                 211

153   <O TO>                   85                83
```

APPROVED

SINGLE ERS

APPENDIX C

TERMINALS VS NON-TERMINALS

1	OPTIONS	98	<RUN>
2	(	99	<COMPILATION>
3	)	100	<OPTIONS>
4	,	101	<OPT LST>

5	<IDENTIFIER>	102	<OPT ITEM>
6	<NUMBER>	103	<SET>
7	-	104	<CONTEXT NAME>
8	CONTEXT	105	<CONTEXT BLK>
9	PROCEDURE	106	<PROG NAME>
10	TASK	107	<PARENT DECL>
11	PARENT	108	<XEQ STAT LST>
12	END	109	<DECL LST>
13	CHARMAP	110	<DEF ITEM>
14	PARAMETER	111	<CHARMAP DEF>
15	TYPE	112	<PARAM DEF>
16	CONSTANT	113	<TYPE DEF>
17	PRESET	114	<CONST DECL>
18	VARIABLE	115	<PRESET DECL>
19	FLAG	116	<VAR DECL>
20	:	117	<FLAG DECL>
21	£	118	<SUBROUTINE DEF>
22	RECORD	119	<PARAMETER>
23	LINK	120	<PARAM DESCRIP>
24	OVERLAY	121	<PARAM ID LST>
25	LIKE	122	<TCOLON>
26	ASSIGNS	123	<EXPRES>
27	RANGE	124	<PARAM IDENTIFIER>
28	TO	125	<TYPE HEAD>
29	STEP	126	<TYPE DESCRIPT>
30	INT	127	<TYPE NAME>
31	SINT	128	<RECORD>
32	STRING	129	<VAR DECL LST>
33	INDEX	130	<LINK>
34	POWERSET	131	<LINK DECL>
35	OF	132	<SCALAR>
36	-	133	<ASSIGNS>
37	>	134	<SCALAR HEAD>
38	ARRAY	135	<IDENTIFIER LST>
39	[	136	<VARIABLE>
40	]	137	<VAR DESCRIP>
41	CHECK	138	<CONSTANT>
42	SUBROUTINE	139	<CON DECL LST>
43	RECYCLE	140	<CON DESCRIP>
44	LEAVE	141	<VCOLON>
45	START	142	<VALUE LST>
46	DEFER	143	<PRESET>
47	WAIT	144	<FLG>
48	SET	145	<FLAG DESCRIP>
49	LOCK	146	<CHECK SPEC>
50	UNLOCK	147	<VALUE GEN>
51	DO	148	<RANGE PART>

52	CALL	149	<RANGE LIST>
53	RETURN	150	<RANGE SPEC>
54	STOP	151	<VALUE PART>
55	PASS	152	<REPLACE>
56	TRANSFER	153	<0 TO>
57	INP	154	<STEP PART>
58	OUT	155	<TO>
59	STATUS	156	<STEP>
60	ENABLE	157	<TYPE>
61	DISABLE	158	<SIMPLE TYPE>
62	OR	159	<ARRAY TYPE>
63	FINISH	160	<PQUALIF>
64	INCR	161	<FAKE ARROW>
65	DECR	162	<QUAL>
66	BY	163	<LEFT SS>
67	.	164	<RIGHT SS>
68	@	165	<SPEC LST>
69	THEN	166	<SPEC ITEM>
70	IF	167	<SUB HEAD>
71	ELSE	168	<XEQ STAT>
72	AND	169	<LAB STAT LST>
73	NOT	170	<GROUP NUMBER>
74	<	171	<BAL STAT>
75	↑	172	<UNBAL IF>
76	BEGIN	173	<SIMPLE XEQ>
77	CASE	174	<BLOCK STAT>
78	NEXT	175	<CASE STAT>
79	CYCLE	176	<CYCLE STAT>
80	WHILE	177	<REP STAT>
81	FOR	178	<IF STAT>
82	+	179	<GROUP NAME>
83	*	180	<PARAMS>
84	/	181	<SOR>
85	\	182	<EVENT>
86	<QSTRING>	183	<SET TO>
87	MIN	184	<LOR>
88	MAX	185	<ENABLE>
89	ASCII64	186	<DISABLE>
90	CONVIS	187	<IO PARAMS>
91	CONVSI	188	<STORE REF>
92	FLAGVAL	189	<PARM C>
93	ALL	190	<CONDITION>
94	ABS	191	<STORE LST>
95	EMPTY	192	<BY PART>

```
 96  *!*
 97  <EMPTY>
```

```
193  <BY>
194  <VAR REF>
195  <EXPRESS PAIR>
196  <VAR>
197  <EXPRES PAIR (>
198  <EXPRES PAIR C>
199  <IF PART>
200  <ELSE>
201  <IF>
202  <COND TERM>
203  <COND FACT>
204  <COND PRIM>
205  <REL OP>
206  <GR OR EQ>
207  <LS OR EQ>
208  <NOT EQ>
209  <BLOCK HEAD>
210  <CASE>
211  <CASE OF>
212  <CYCLE HEAD>
213  <QUALIFIER>
214  <CONTROL PART>
215  <FOR PART>
216  <TERM>
217  <PRIMARY>
```

A DESCRIPTION OF SINGLE:

THE SINGER IMPLEMENTATION LANGUAGE

NOVEMBER 1974

1. INTRODUCTION

THE SINGLE LANGUAGE IS DESIGNED TO BE SUITABLE AS AN IMPLEMENTATION LANGUAGE FOR SYSTEMS PROGRAMMING OF THE SYSTEM 10, MODEL 20 & 21, SYSTEM 11 DPU AND MPU. CHOICE OF OBJECT CODE WILL BE OBTAINED BY A COMPILER DIRECTIVE PRECEDING THE SOURCE CODE.

A. OBJECTIVES

THE PARTICULAR OBJECTIVES DERIVED FROM THE ABOVE GENERAL OBJECTIVE ARE:

1. HIGH LEVEL DESCRIPTION OF LOGIC AND DATA.
2. MODULAR CONSTRUCTION OF PROGRAMS.
3. MINIMAL MACHINE AND DEVICE-DEPENDENT CODE.
4. MINIMAL REQUIREMENTS FOR RUN-TIME SUPPORT OF OBJECT CODE.
5. CONDITIONAL COMPILATIONS FOR DIFFERENT PROCESSORS AND CONFIGURATIONS.
6. INDEPENDENTLY COMPILABLE MODULES.
7. MAXIMAL READABILITY OF SOURCE CODE.

8. SUPPORT FOR PROGRAM STRUCTURES FOUND TO BE USEFUL ON SBM EQUIPMENT, E.G., OVERLAYS.
9. SPECIAL CONSTRUCTS TO HELP OBTAIN OBJECT CODE EFFICIENCY.

B. SOURCES

THE PRIMARY SOURCES FOR THE LANGUAGE PROPOSED ARE PASCAL, SUE, (A DESCENDANT OF PASCAL) AND BURROUGHS B1700 IMPLEMENTATION LANGUAGE. MANY FEATURES OF THESE LANGUAGES HAVE BEEN OMITTED, PRIMARILY TO MINIMIZE RUN-TIME SUPPORT REQUIREMENTS AND/OR MACHINE DEPENDENCIES. THE RESULT IS A COLLECTION OF CONCENSUS COMPROMISES, AND NEW IDEAS (A FEW) WHICH SEEM BEST SUITED TO SBM'S NEEDS.

2. LANGUAGE SUMMARY

A. GENERAL FEATURES

THE SINGLE LANGUAGE DIFFERS FROM MANY HIGH LEVEL PROGRAMMING LANGUAGES IN THAT IT IS AIMED AT THE SKILLED PROFESSIONAL PROGRAMMER. THERE IS MORE THAN USUAL EMPHASIS ON SYSTEM-LEVEL INTEGRATION OF PROGRAMS, ON CODE EFFICIENCY, ON CAREFUL DESIGN AND PRECISE SPECIFICATION OF DESIGN. A FURTHER FEATURE IS A COMPILER CONTROLLED SOURCE OUTPUT LISTING WHICH REFORMATS SOURCE INPUT INTO A STANDARDIZED STRUCTURE WHICH ENHANCES READABILITY VIA CONTEXT - DEPENDENT INDENTATION, TABBING, ETC.

OTHER FEATURES WORTH NOTING ARE:

- SEPARATE SECTIONS FOR SPECIFYING

  COMPILE TIME PARAMETERS \
  CONSTANTS > (ONLY AREAS IN SOURCE WHERE LITERAL VALUES MAY APPEAR)
  PRESET VARIABLES /
  NON-PRESET VARIABLES
  FLAGS (FOR COOPERATING PROCESSES)

- COMPILE TIME EVALUATION OF PARAMETRIC AND CONSTANT PARTS OF

EXPRESSIONS AND CONDITIONS, WITH CONDITIONS FULLY EVALUATED AND BRANCH SELECTED AT COMPILE TIME, WHEN THE CONDITION CONTAINS NO RUN-TIME VARIABLES.

- TWO TYPES OF PROCEDURAL MODULES: 'TASK' AND 'PROCEDURE', ALLOWING CHAINED OVERLAYS AND ASYNCHRONOUS PROCESSING.

- NESTED CONTROL STRUCTURES, NO 'GO-TO' CONSTRUCT.

IN GENERAL, THERE ARE THREE KINDS OF STATEMENTS IN THE SINGLE LANGUAGE: DEFINITIONS, DECLARATIONS AND ACTIONS. THE NEXT THREE SECTIONS SUMMARIZE THE FEATURES OF EACH KIND OF STATEMENT, AND THE FOLLOWING SECTION DISCUSSES COMPILATION STRUCTUR

IN ALL OF THE MATERIAL WHICH FOLLOWS, TWO NOTATIONAL ODDITIES WILL BE FOUND. THE APPEARANCE OF

<.....>

INDICATES THAT THE SOURCE LANGUAGE INPUT CONTAINS AN INSTANCE OF THE SYNTACTIC CONSTRUCT NAMED BY ....
THE APPEARANCE OF

(*........*)

MEANS THAT THE ENCLOSED ELEMENT OF SOURCE LANGUAGE IS OPTIONAL

B. DEFINITION STATEMENTS

THE DEFINITION STATEMENTS IN SINGLE EXPRESS INFORMATION AFFECTING SYSTEM STRUCTURE. THEY DESCRIBE DATA STRUCTURES, PROGRAM STRUCTURES AND PARAMETER VALUES GOVERNING PARAMETRIC AND CONDITIONAL COMPILATIONS. THE SPECIFIC KINDS OF DEFINITION STATEMENTS ARE:

• < CONTEXT / PROCEDURE / TASK >	<IDENT>	DESCRIBES WHICH TYPE OF SOURCE BLOCK FOLLOWS AND GIVES A NAME FOR THE BLOCK.
• PARENT	<IDENT>	NAMES THE NEXT HIGHER LEVEL BLOCK GOVERNING THIS BLOCK.
• CHARMAP <DEF>		DEFINES FUNCTIONS MAPPING CHARACTERS ONTO THE INTEGERS. THESE MAY BE USED

```
            <DEF>                   FOR CODE CONVERSIONS, SCANNING, ETC.
              :
          END CHARMAP

      .   PARAMETER                 DEFINES VALUES FOR SYMBOLS APPEARING
            <DEF>                   WITHIN THIS AND LOWER LEVEL BLOCKS
            <DEF>                   WHICH ARE TO FUNCTION AS COMPILE TIME
              :                     PARAMETERS.
          END PARAMETER

      .   TYPE                      DEFINES NAMES WHICH SPECIFY PARTICULAR
            <DEF>                   KINDS OF DATA STRUCTURING.  WITHIN
            <DEF>                   THE DECLARATION STATEMENTS VARIABLES
              :                     WILL BE DECLARED AS POSSESSING THE
          END TYPE                  TYPES DEFINED HERE.  NAMES ALL BLOCKS
                                    DIRECTLY SUBORDINATE TO THIS BLOCK,
                                    SPECIFIES THE ARGUMENTS USED BY EACH.
```

C. DECLARATION STATEMENTS

THE DECLARATION STATEMENTS NAME AND DESCRIBE THE ITEMS OF DATA USED WITHIN THE PROGRAM, MAKING USE OF THE TYPES AND PARAMETERS GIVEN IN THE DEFINITIONS.

WHERE DATA ITEMS ARE TO HAVE CONSTANT VALUES, THE DECLARATION FORM IS:

```
CONSTANT
  <VALUEDEC>
  <VALUEDEC>
    :
END CONSTANT
```

THE FORM OF EACH <VALUEDEC> IS

```
  <IDENT>      :<TYPE>  :<VALUE>,<VALUE>,..
```

WHERE THE <VALUE>S ARE DIRECT STATEMENTS OF THE VALUE FOR EACH COMPONENT OF THE DATA ITEM <IDENT>.

THE <TYPE> MAY BE A TYPE NAMED IN THE DEFINITION SECTION OR ONE OF A SET OF PREDEFINED TYPES BUILT INTO THE COMPILER.

IN THE CASE OF VARIABLES WITH PRESET INITIAL VALUES, THE FORM

IS

PRESET
<PREDEC>
<PREDEC>
:
END PRESET

FOR DATA ITEMS WHICH ARE VARIABLE AND HAVE NO PRESET VALUE, THE DECLARATION FORM IS:

VARIABLE
<DECLAR>
<DECLAR>
:
END VARIABLE

EACH <DECLAR> IS OF THE FORM
<IDENT1>, <IDENT2>,....
....
.....<IDENTN> :<TYPE>

WHERE THE <IDENTI> ARE THE NAMES OF VARIABLES HAVING THE COMMON <TYPE> INDICATED.

IF ANY VARIABLES ARE TO BE USED AS INTERLOCKS AMONG COPROCESSES (ASYNCHRONOUSLY OPERATING PROGRAMS) THEY ARE DECLARED BY

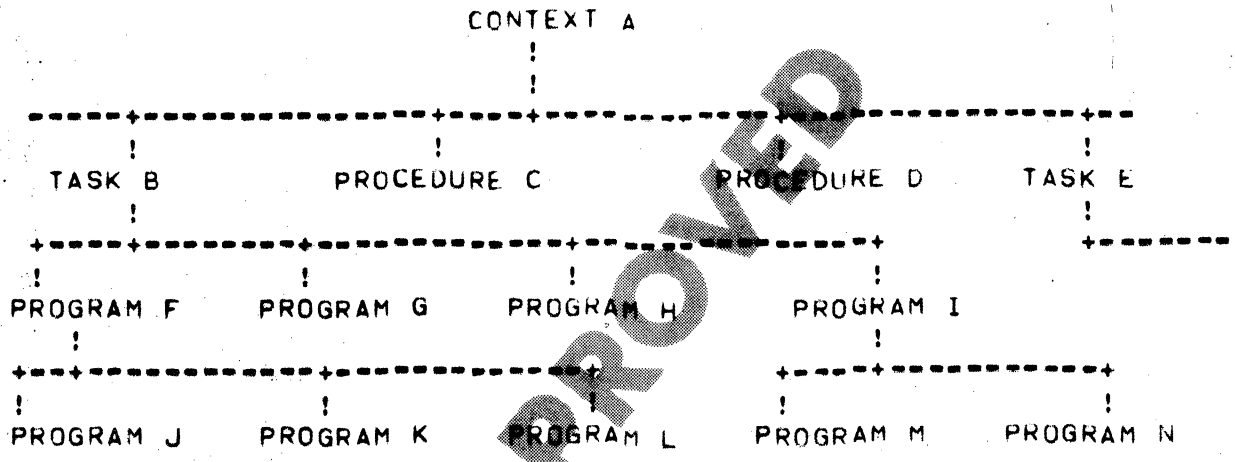
FLAG
<IDENT>
<IDENT>
:
END FLAG

D. ACTION STATEMENTS

THE ACTION STATEMENTS DESCRIBE THE PROGRAM EXECUTION: THE MANIPULATING OF DATA AND FLOW OF CONTROL, AND CONSTITUTE WHAT IS COMMONLY CALLED THE 'EXECUTABLE PROGRAM'.

E. COMPILATION STRUCTURE

A COMPLETE SOURCE PROGRAM CONSISTS OF A SET OF NAMED BLOCKS. THERE IS A UNIQUE NAMED CONTEXT BLOCK WHICH CONTAINS ONLY DEFINITIONS AND DECLARATIONS, INCLUDING A 'LINK' STATEMENT TO ONE OR MORE TASK OR PROCEDURE BLOCKS. THE TYPICAL PRECEDENCE STRUCTURE IS:



HERE TASK B IS THE HIGHEST LEVEL BLOCK OF A COMPLETE PROGRAM. TASK E IS THE HIGHEST LEVEL BLOCK OF ANOTHER COMPLETE PROGRAM WHICH MAKES USE OF THE SAME CONTEXT. PROCEDURES C AND D ARE ACCESSIBLE TO ANY PROGRAM USING CONTEXT A, AND WILL BE LOADED AS PART OF ANY SUCH PROGRAM. A BLOCK IS CALLED THE PARENT OF ANY BLOCK LINKED TO IT AT THE NEXT LOWER LEVEL. THUS, FOR EXAMPLE, F IS THE PARENT OF J,K, AND L. J, K, AND L ARE SAID TO BE DIRECTLY SUBORDINATE TO F. IF TWO BLOCKS, SUCH AS B AND K, ARE SUCH THAT THERE EXISTS A SEQUENCE OF BLOCKS B,X,Y,..K WITH EACH BLOCK THE PARENT OF THE FOLLOWING BLOCK, THEN K IS SAID TO BE SUBORDINATE TO B. FOR ANY BLOCK, THE SCOPE OF THE BLOCK IS DEFINED AS THE BLOCK ITSELF PLUS ALL BLOCKS WHICH ARE SUBORDINATE TO IT.

ANY BLOCK MAY BE ALTERED AND RECOMPILED WITHOUT AFFECTING ANY BLOCKS NOT SUBORDINATE, AS LONG AS ITS NAME AND ARGUMENTS ARE UNCHANGED. A CHANGE IN A BLOCK MAY REQUIRE RECOMPILATION OF ALL SUBORDINATE PROGRAM BLOCKS. THE COMPILATION OF A PROGRAM OR CONTEXT BLOCK PRODUCES A SYMBOL TABLE WHICH REPRESENTS THE DEFINITIONS AND DECLARATIONS OF THAT BLOCK. ALSO, COMPILATION OF A CONTEXT BLOCK PRODUCES A RELOCATABLE DECK OF

THE INITIALIZED DATA DECLARED WITHIN IT. THE COMPILATION OF A PROGRAM BLOCK ALSO PRODUCES A RELOCATABLE DECK CONTAINING BOTH DATA AND OBJECT CODE.

A PROGRAM BLOCK MAY CONTAIN ALL THREE KINDS OF STATEMENTS.

A PROGRAM BLOCK MAY REFERENCE ONLY DATA DECLARED WITHIN ITSELF OR WITHIN SOME BLOCK TO WHICH IT IS SUBORDINATE. IF PROGRAM X CONTAINS A LINK TO PROGRAM Y THEN PROGRAM Y MAY BE INVOKED ONLY BY PROGRAMS IN THE SCOPE OF X BUT NOT IN THE SCOPE OF Y.

3. IDENTIFIERS AND VALUES

IDENTIFIERS SERVE TO DENOTE DATA, TYPES, BLOCKS, STATEMENT GROUPS, SUBROUTINES, AND VALUES OF SCALAR TYPE VARIABLES. THEIR DENOTATION MUST BE UNIQUE WITHIN THE SCOPE OF THEIR VALIDITY, I.E., WITHIN THE BLOCK IN WHICH THEY ARE SPECIFIED AND ALL SUBORDINATE BLOCKS. AN IDENTIFIER IS 1 TO 10 CHARACTERS, THE FIRST OF WHICH IS ALPHABETIC AND THE REST OF WHICH ARE ALPHABETIC, DIGITS, OR THE UNDERLINE (←). IN THE CASE OF DATA WHICH IS A PARAMETER, A SPECIAL PARAMETER IDENTIFIER IS USED, CONSISTING OF 2 TO 10 CHARACTERS BEGINNING WITH '£', FOLLOWED BY AN ALPHABETIC, FOLLOWED BY ALPHABETICS, DIGITS OR UNDERLINE.

EXAMPLES:

```
SAM              IDENTIFIERS
A←1←3←C

£K←ONE           PARAMETER IDENTIFIER
```

LITERAL OR SELF-DEFINING VALUES FOR DATA ITEMS APPEAR IN PARAMETER DEFINITIONS AND IN THE VALUE SPECIFICATIONS OF CONSTANT AND PRESET DECLARATIONS ONLY. THESE VALUES ARE OF TWO KINDS, EITHER (NON-NEGATIVE) INTEGERS OR CHARACTER STRINGS. INTEGER VALUES ARE REPRESENTED BY A SEQUENCE OF AT MOST TEN DECIMAL DIGITS.

EXAMPLES:

```
PARAMETER
```

```
        .
        .
        .
    £BUFSIZE        :32
        .
        .
    END PARAMETER
    PRESET
        .
        .
    BASE            :INT[£MAXB+£MAXC]   :50
    LINK            :INDEX->V           :2
        .
        .
    END PRESET
```

CHARACTER STRING VALUES ARE REPRESENTED BY THE CHARACTERS THEMSELVES, ENCLOSED WITHIN THE BOUNDING DELIMITER APOSTROPHE (') , OR WITHIN BOUNDING QUOTES (") . UP TO 60 CHARACTERS MAY BE GIVEN IN THIS MANNER. EXAMPLE:

```
    CONSTANT
        .
        .
    A               :STRING[£K0]        :" "   (IN THIS INSTANCE,
                                                A IS FILLED WITH
                                                BLANKS)
    B               :STRING[£K3]        :'SAM' (IN THIS INSTANCE,
                                                B IS FILLED WITH
                                                'SAM' PLUS £K3-3
                                                BLANKS IF £K3>3.
                                                IF £K3<3, B IS FILLED
                                                WITH £K3 CHARACTERS
                                                FROM LEFT, AND ERROR
                                                MESSAGE GENERATED.
    END CONSTANT
```

IF THE DELIMITER "'" IS TO BE A PART OF A STRING VALUE, THE QUOTE '"' IS USED AS A DELIMITER AND CONVERSELY.

IN THE CASE WHERE A STRING VALUE IS LONGER THAN 60 CHARACTERS, OR WHERE BOTH DELIMITERS ARE USED WITHIN THE STRING, THE PLUS SIGN FOR STRING CONCATENATION MAY BE USED:

```
    PRESET
        .
        .
```

```
        A           :STRING[£KTST]    :'ABCDEFGHIJKLMNOPQRST'+
                                       'UVWXYZ                '+
                                       '0123456789            '+
                                       '+-/\ *,.?+"           '
        B           :STRING[£KTST]    :'"IS'+"'NT LEGAL HERE"
      END PRESET
```

A PARTICULAR TYPE OF VARIABLE, THE SCALAR VARIABLE, TAKES ON VALUES WHICH ARE INDICATED SYMBOLICALLY BY GIVING NAMES FOR EACH OF THE POSSIBLE VALUES. THESE SYMBOLIC VALUES MAY APPEAR IN ANY PART OF A PROGRAM. EXAMPLE:

```
    TYPE
      .
      .
    DIRECT = (NORTH, SOUTH, EAST, WEST)
      .
    END TYPE
      .
    VARIABLE
      .
      .
      A       :DIRECT
      .
    END VARIABLE
      .
    IF A = EAST THEN...
```


WHERE VALUES ARE TO BE ASSIGNED TO ARRAY TYPE VARIABLES, A VALUE LIST IS USED, CONSISTING OF A SEQUENCE OF VALUE GENERATORS SEPARATED BY COMMAS. A VALUE GENERATOR IS EITHER A VALUE OF THE KINDS ALREADY DESCRIBED, OR IT MAY BE OF THE FORM

```
  RANGE:=(*<PX1>*) TO <PX2>(*STEP <PX3>*):<I-EXP>
```

WHERE <PXN> IS A PARAMETRIC EXPRESSION EVALUATING TO A NON-NEGATIVE INTEGER AT COMPILE TIME AND <I-EXP> IS AN EXPRESSION INVOLVING VALUES, PARAMETERS, PREVIOUSLY ESTABLISHED CONSTANT OR PRESET DATA ITEMS, AND THE RUNNING PARAMETER RANGE. DEFAULT VALUES OF <PX1> AND <PX3> ARE 0 AND 1 RESPECTIVELY.
EXAMPLE:

```
  PRESET
    .
   LINKS      :ARRAY [£Z] OF INDEX->POOL :RANGE:= TO 19
     :2*RANGE+5
```

```
      .
    END PRESET
```

GENERATES THE VALUES 5, 7, 9, ..., 43.

AS ANOTHER EXAMPLE:

```
  PRESET
   V1        :INT[£A]                    :5
   V2        :INT[£A]                    :£A-1
    .
   A         :ARRAY [£A] OF INT[£KL]     :0,1,2,3,5,
      RANGE:= V1 TO V2 :99,0
```

PRESETS THE ELEMENTS OF ARRAY A TO THE VALUES 0, 1, 2, 3, 5, 99, 99, ..., 99, 0.

NOTE THAT 'RANGE' IS A RESERVED WORD AND MAY NOT BE USED AS AN IDENTIFIER FOR ANY OTHER PURPOSE.

4. DATA

A. DATA CLASSES

DATA MAY BE OF SEVERAL CLASSES:

- PARAMETERS
- CONSTANTS
- PRESETS
- VARIABLES
- FLAGS

AND ANY, EXCEPT PARAMETERS, MAY HAVE ATTRIBUTES 'CHECK' 'F', 'S' FOR COMPILER CHECKING OF FETCHES AND/OR STORES OF THE ITEM. PARAMETERS ARE COMPILE-TIME QUANTITIES EVALUATED BY THE COMPILER FOR PARAMETRIC AND CONDITIONAL COMPILATION. CONSTANTS ARE ALSO KNOWN AT COMPILE TIME AND USED BY THE COMPILER, BUT ARE ALSO AVAILABLE TO THE RUN-TIME PROGRAM (I.E., LOADED WITH THE PROGRAM). AN ACTION STATEMENT WHICH ASSIGNS A VALUE TO A PARAMETER OR CONSTANT DATA ITEM GENERATES A COMPILE TIME DIAGNOSTIC. PRESETS ARE TREATED LIKE CONSTANTS IN COMPILE TIME EXPRESSIONS BUT NOT IN RUN TIME EXPRESSIONS, WHERE THEY ARE TREATED LIKE VARIABLES. THEIR VALUES ARE LOADED WITH THE PROGRAM.

VARIABLES SHOULD HAVE A VALUE ASSIGNED TO THEM BY THE OBJECT PROGRAM BEFORE BEING EVALUATED WITHIN AN EXPRESSION, SINCE THE COMPILER DOES NOT INITIALIZE VALUES FOR THEM AT LOAD TIME.

FLAGS ARE USED BY COOPERATING PROCESSES AND MUST BE ATTACHED VIA A 'LOCK' OPERATION BEFORE READING OR WRITING. FLAGS MAY HAVE ADDITIONAL CHECK ATTRIBUTES 'L,''U' FOR ACTIONS WHICH LOCK OR UNLOCK THEM.

B. SIMPLE TYPES

DATA WHICH MAY HAVE ONLY ONE VALUE AT ANY GIVEN TIME ARE CALLED SIMPLE TYPES. THE COMPILER RECOGNIZES FOUR BUILT-IN SIMPLE TYPES:

- INT [<NVAL>] WHERE <NVAL> IS A COMPILE TIME EXPRESSION CONTAINING NO VALUES, EVALUATING TO A NON-NEGATIVE INTEGER. A VARIABLE OF THIS TYPE MAY TAKE ON INTEGER VALUES N WITH 0 <= N <= (VALUE OF <NVAL>). THUS IF THE PARAMETER £A HAS THE VALUE 99, THEN A VARIABLE OF TYPE INT[£A] MAY TAKE ON ANY OF THE VALUES 0,1,2,...,99. THE MAXIMUM SIZE OF AN INT VARIABLE IS TEN DECIMAL DIGITS.

NOTE: Y:INT[£B] AND X:INT[£C] ARE DIFFERENT TYPES UNLESS PARAMETERS £B AND £C ARE GIVEN THE SAME VALUE AT COMPILE TIME.

- SINT [<NVAL>], MEANING A VARIABLE OF TYPE INT[<NVAL>] BUT ALSO CARRYING AN ALGEBRAIC SIGN SO THAT IT MAY TAKE ON VALUES -<NVAL>,-<NVAL>+1,...,-1,0,1,2,..,<NVAL>-1, <NVAL>.

- INDEX->IDENT WHERE IDENT IS THE IDENTIFIER OF AN ARRAY. IF N IS THE DECLARED MAXIMAL NUMBER OF ELEMENTS IN THE ARRAY, THEN THE INDEX VARIABLE IS LOGICALLY EQUIVALENT TO A VARIABLE OF TYPE INT[N]. HOWEVER, IF THE PRIMARY USE OF THE VARIABLE IS AS A SELECTOR OF ELEMENTS OF THE ARRAY THEN MORE EFFICIENT CODE WILL RESULT IF THE VARIABLE IS DECLARED TYPE INDEX.

W:INDEX->A AND Z:INDEX->B ARE DIFFERENT TYPES EVEN THOUGH ARRAYS A AND B MAY HAVE IDENTICAL STRUCTURE. Z MAY BE USED AS A SUBSCRIPT FOR A, BUT THE RESULT IS INEFFICIENT.

- FLAG. A FLAG IS IMPLICITLY DECLARED OF TYPE INT [99]

AND PRESET TO ZERO. A FLAG MAY APPEAR IN AN EXPRESSION ONLY AS THE ARGUMENT OF THE FLAGVAL FUNCTION.

THE OTHER SIMPLE TYPES ARE CALLED SCALAR TYPES. THEY ARE DEFINED TYPES, WHOSE DEFINITION CONSISTS OF NAMING THE TYPE, AND LISTING THE NAMES OF ALL VALUES SUCH A VARIABLE MAY ASSUME:

<SCALAR TYPE> = (<IDENT1>,<IDENT2>,...,<IDENT>)

EXAMPLE OF A SCALAR TYPE DEFINITION:

WEEKDAY = (MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY)

THUS, ANY VARIABLE DECLARED OF TYPE WEEKDAY CAN TAKE ON ONE OF THE FIVE POSSIBLE VALUES LISTED. NOTE THAT THESE ARE SYMBOLIC DEFINITIONS OF INDIVIDUAL VALUES RATHER THAN LITERAL REPRESENTATIONS.

C. COMPOUND TYPES

- STRING [<NUMCHAR>] WHERE <NUMCHAR> IS A COMPILE TIME EXPRESSION CONTAINING NO VALUES, EVALUATING TO A NON-NEGATIVE INTEGER. A VARIABLE OF TYPE STRING [£A] HAS A VALUE WHICH IS AN ORDERED STRING OF CHARACTERS C:

  C0,C1,..,CN (WHERE £A HAS VALUE N+1).

  ANY SUBSTRING OF SUCCESSIVE CHARACTERS WITHIN SUCH A VARIABLE X MAY BE REFERENCED BY THE SUBSTRING NOTATION

  X(<INIT>) OR
  X(<INIT>,<NUMCHAR>), WHERE <INIT> AND <NUMCHAR> ARE

  RUN-TIME EXPRESSIONS EVALUATING TO NON-NEGATIVE INTEGERS. <INIT> SPECIFIES THE ORDINAL POSITION OF THE LEFT-MOST CHARACTER OF THE SUBSTRING AND <NUMCHAR> IS THE NUMBER OF CHARACTERS IN THE SUBSTRING STARTING WITH <INIT>. IF <NUMCHAR> IS OMITTED, THE DEFAULT VALUE IS 1 CHARACTER. THE MAXIMUM LENGTH ALLOWED FOR A STRING DATA ITEM IS 2048 CHARACTERS.

- RECORD. THIS IS A DEFINED TYPE WHOSE DEFINITION NAMES

THE RECORD TYPE AND LIST NAMES AND TYPES OF THE FIELDS WITHIN THE RECORD. AN EXAMPLE OF A RECORD TYPE DEFINITION IS

```
BUFFER = RECORD
  FLINK,BLINK          :INDEX->SPACE
  NUMELS               :INDEX->ITEMS
  ITEMS                :ARRAY [£SIZE] OF INT[£N]
END
```

THUS, A VARIABLE OF TYPE BUFFER CONTAINS SEVERAL COMPONENTS: TWO INDICES FLINK & BLINK OF AN ARRAY SPACE, A COUNTER NUMELS (WHICH MIGHT COUNT THE NUMBER OF ELEMENTS PRESENTLY STORED IN THE ARRAY ITEMS) AND THE ARRAY ITEMS. FIELDS WITHIN A RECORD ARE REFERENCED BY QUALIFICATION, E.G., OLDBUF.BLINK, WHERE OLDBUF IS DECLARED OF TYPE BUFFER. THE TYPE OF A RECORD FIELD MAY ITSELF BE A RECORD.

- ARRAY [<NUMEL>] OF <TYPE>, WHERE <NUMEL> IS A COMPILE TIME EXPRESSION CONTAINING NO VALUES, EVALUATING TO A NON-NEGATIVE INTEGER. AN ARRAY IS AN ORDERED SET OF COMPONENTS EACH OF THE SAME <TYPE>, WITH A MAXIMUM NUMBER OF COMPONENTS DETERMINABLE AT COMPILE TIME. A COMPONENT OF AN ARRAY IS REFERENCED BY SUBSCRIPTING, VIA AN INT OR INDEX TYPE EXPRESSION, OF THE ARRAY NAME. EXAMPLE:

```
SPACE    :ARRAY[£NUMBUF] OF BUFFER
```

REFERENCE TO AN ARRAY COMPONENT MAY BE, E.G.

```
NEWBUF:=SPACE[CURBUF.FLINK]      OR
A[I+J]:=A[I]+A[I-J]
```

NOTICE THAT ARRAYS OF ARRAYS ARE NOT ALLOWED.

- POWERSET. A POWERSET STRUCTURE IS A SET OF VALUES WHICH ARE THE SUBSETS OF THE SET OF VALUES OF ITS BASE TYPE, WHICH IS A DEFINED SCALAR TYPE. FOR EXAMPLE:

```
COMPONENT = (BEGIN,FREE)
  .
  .
  .
STATE      :POWERSET OF COMPONENT
```

THE VARIABLE STATE TAKES ON FOUR POSSIBLE VALUES, CORRESPONDING TO THE PRESENCE OR ABSENCE OF EACH OF THE POSSIBLE VALUES OF COMPONENT. ANOTHER EXAMPLE: IF

```
PRIMARY = (RED,YELLOW,BLUE)
  .
  .
  .
HUE      :POWERSET OF PRIMARY
```

THEN THE HUE CAN TAKE ON ANY OF EIGHT POSSIBLE VALUES CORRESPONDING TO THE PRESENCE OR ABSENCE OF EACH POSSIBLE VALUE OF PRIMARY.

FOR ANY POWERSET VARIABLE, TWO SPECIAL SYMBOLIC VALUES ARE ALLOWED: EMPTY AND ALL, WITH THE OBVIOUS MEANINGS.

THE MAXIMUM NUMBER OF COMPONENTS OF A SCALAR FROM WHICH A POWERSET IS DERIVABLE IS 16.

5. DEFINITIONS

THE FIRST PORTION OF ANY SOURCE BLOCK IS THE DEFINITION SECTION IT CONTAINS UP TO FIVE SUBSECTIONS IN THE FOLLOWING ORDER:

	CONTEXT BLOCK	PROGRAM BLOCK
BLOCK HEAD	REQUIRED	REQUIRED
PARENT	INVALID	REQUIRED
CHARMAP	ALLOWED	ALLOWED
PARAMETER	ALLOWED	ALLOWED
TYPE	ALLOWED	ALLOWED

A. BLOCK HEAD. THIS STATEMENT IS OF THE FORM

```
CONTEXT
PROCEDURE    <IDENT>
TASK
```

<IDENT> IS THE NAME OF THE BLOCK HEADED BY THIS STATEMENT.

B. PARENT. THIS STATEMENT IS OF THE FORM

```
PARENT <IDENT>
```

<IDENT> NAMES THE UNIQUE BLOCK WHICH HAS DECLARED THIS BLOCK TO BE IN ITS SCOPE.

C. CHARMAP. THIS CONSTRUCT DEFINES A NAMED MAPPING OF CHARACTERS INTO INTEGERS, WHICH MAY THEN BE USED TO EXPRESS VARIOUS TYPES OF CHARACTER-DEPENDENT LOGIC SUCH AS ACTION TABLES. EXAMPLE:

```
CHARMAP
    .
  OPSCAN = (
    0:'+'
    1:'-'
    2:'*'
    3:'/.,='
    4:OTHER
      )
```


THE EXPRESSION OPSCAN(LINE[I]) EXAMINES THE ITH CHARACTER OF THE STRING NAMED LINE AND GENERATES AN INTEGER VALUE BETWEEN 0 AND 4 DEPENDING ON THE CHARACTER FOUND. THE VALUES PRECEDING THE COLON ARE DERIVED FROM POSITION ONLY. IF THE COLON IS NOT FOLLOWED BY 'OTHER', THEN THE FIRST CHARACTER AFTER THE COLON IS A DELIMITER. ALL CHARACTERS FOUND BEFORE THE NEXT OCCURRENCE OF THAT DELIMITER MAP INTO THE SAME VALUE.

THE COMPILER CONTAINS A BUILT-IN CHARMAP CALLED ASCII64 WITH IMPLICIT DECLARATION

```
  0:' '
  1:'!'
  2:'"'
  .
  .
  .
 16:/0/
 17:#1$
  .
  .
  .
 32:7A7
  .
```

```
        .
        .
      63:Z-Z
      64:OTHER
        )
```

D. PARAMETER. THIS STATEMENT IS OF THE FORM

```
    PARAMETER
      <DEFINITION>
      <DEFINITION>
        .
        .
    END
```

EACH <DEFINITION> IS OF THE FORM

```
    <PARAMIDENT>     :<VALUE EXPRESSION>
```

WHERE <PARAMIDENT> IS 2 TO 10 CHARACTERS BEGINNING WITH £, FOLLOWED BY AN ALPHABETIC, FOLLOWED BY 0-8 ALPHABETIC, DIGIT OR UNDERLINE CHARACTERS. THE <VALUE EXPRESSION> MUST YIELD A NON-NEGATIVE INTEGER VALUE, OR A STRING VALUE. ALL PARAMETER NAMES USED IN THE VALUE EXPRESSION MUST HAVE BEEN DECLARED, AND, HENCE EVALUATED PRIOR TO THIS PARAMETER DEFINITION. ALL REFERENCES TO DATA ITEMS MUST BE TO CONSTANT OR PRESET ITEMS DECLARED IN SOME HIGHER LEVEL BLOCK.

```
    PARAMETER
      £BUFSIZE          :20
      £CONFIG           :MAX(£BUFSIZE,8*£INCREM)
      £INCREM           :3
    END
```

THIS SEQUENCE OF DEFINITIONS IS INVALID, SINCE £INCREM IS REFERENCED BEFORE BEING DECLARED.

E. TYPE. THIS STATEMENT GIVES DEFINED TYPES USED IN THE SCOPE OF THIS BLOCK AND NOT DEFINED IN SOME HIGHER LEVEL BLOCK. THE FORM IS

```
    TYPE
      <DEF>
      <DEF>
        .
        .
    END
```

THE FORM OF EACH <DEF> IS

E1) IDENTO = (IDENT1,IDENT2,...,IDENTN)

FOR DEFINING SCALAR TYPES, OR

E2) <IDENTO> = RECORD

```
    <IDENT1>,<IDENT2>,...,<IDENTN>     :<TYPE0>
    <IDENTN+1>,...                     :<TYPE1>
      .
      .
  END <IDENTO>
```

FOR RECORD TYPES. HAVING DEFINED A RECORD TYPE NAMED IDENTO, ONE MAY GIVE A DECLARATION IN THE DECLARATION SECTION DEFINING A VARIABLE IDENTK TO BE TYPE IDENTO. FIELDS WITHIN THIS RECORD ARE REFERENCED BY GIVING THE RECORD NAME FOLLOWED BY A PERIOD, FOLLOWED BY THE NAME OF THE PARTICULAR FIELD GIVEN IN THE DEFINITION. EXAMPLE:

```
TYPE
  .
  .
  .
 A = RECORD
  F1          :STRING[&KL]
  F2          :ARRAY[1..&KMAX] OF INT[&SIZE]
 END A
  .
END TYPE
  .
VARIABLE
  .
  B             :A
  .
END VARIABLE
  .
  Q:=B.F2[K+J]
```

E3) <IDENT> = / TASK \ / LINK \
\ PROCEDURE / \ OVERLAY /

```
    <IDENT1>,<IDENT2>,... :<TYPE>,
        .
        .
```

```
        .
    ASSIGNS
     <IDENTM>,<IDENTN>,...   :<TYPEJ>
        .
        .
    END
```

THE LINK AND OVERLAY DEFINITIONS NAME EACH BLOCK WHICH IS DIRECTLY SUBORDINATE TO THE CURRENT BLOCK, AND IN ADDITION THEY SPECIFY THE KIND OF BLOCK (TASK OR PROCEDURE) AND THE ARGUMENTS, IF ANY. THE TYPES IN THE OVERLAY OR LINK DEFINITIONS INDICATE THE TYPES OF THE ARGUMENTS. THE EFFECT IS TO DEFINE A RECORD TYPE HAVING THE SAME NAME AS THE PROGRAM. INVOKING A PROGRAM INCLUDES NAMING A VARIABLE WHICH HAS BEEN DECLARED TO BE OF THAT TYPE. THE 'ASSIGNS' INDICATE WHICH OF THE FIELDS ARE ASSIGNED VALUES BY THE INVOKED PROGRAM WHEN IT IS EXECUTED. THUS FIELDS NAMED PRIOR TO THE 'ASSIGNS' ARE INPUT ARGUMENTS, THOSE WHICH FOLLOW THE 'ASSIGNS' ARE BOTH INPUT AND OUTPUT ARGUMENTS.

6. DECLARATIONS

DATA DECLARATIONS SERVE TO ASSOCIATE AN IDENTIFIER WITH A DATA ITEM OF A SPECIFIC TYPE, TO RESERVE SPACE FOR THE ITEM, AND IN SOME CASES TO ATTACH A VALUE TO THAT ITEM. ALL DATA ITEMS MUST BE DECLARED, AND ARE USABLE ONLY WITHIN THE SCOPE OF THE BLOCK IN WHICH THEY ARE DECLARED. THE DECLARATIONS SECTION MAY APPEAR IN A CONTEXT OR PROGRAM BLOCK, AND CONTAINS FOUR SUBSECTIONS, EACH OPTIONAL, IN THE FOLLOWING ORDER:

```
   CONSTANTS
   PRESETS
   VARIABLES
   FLAGS
```

TO BEGIN THE DATA DECLARATIONS, ANY CONSTANT (READ-ONLY) DATA IS DECLARED IN THE CONSTANT SECTION:

```
  CONSTANT
    <VALUEDEC>
    <VALUEDEC>
       .
       .
  END CONSTANT
```

THE FORM OF EACH <VALUEDEC> IS

```
<IDENT>   :<TYPE> (* CHECK CLAUSE *)
           :<VALUE GEN>,<VALUE GEN>,...<VALUE GEN>
```

VALUES CAN BE DECLARED ONLY FOR DATA-ITEMS OF TYPE INT, INDEX AND STRING, OR FOR ARRAYS OF SUCH ITEMS. THUS MORE THAN ONE VALUE IN A CONSTANT OR PRESET DECLARATION IS POSSIBLE ONLY FOR AN ARRAY TYPE. IN THIS CASE THE NUMBER OF VALUES MUST EQUAL THE DIMENSION OF THE ARRAY.

AFTER THE CONSTANT SECTION, ANY CHANGEABLE DATA WHICH IS TO HAVE VALUES UPON INITIATION OF THE ASSOCIATED PROGRAM IS DECLARED IN THE PRESET SECTION:

```
PRESET
  <VALUEDEC>
  <VALUEDEC>
     .
     .
END PRESET
```

WHERE EACH <VALUEDEC> IS AS IN THE CONSTANT SECTION.

THE FORM OF THE VARIABLES SECTION IS:

```
VARIABLE
  <DECLAR>
  <DECLAR>
     .
     .
     .
END VARIABLE
```

WHERE EACH <DECLAR> IS OF THE FORM

```
<IDENT0>,<IDENT1>,...,<IDENTI>,

<IDENTJ>,<IDENTK>                :<TYPE>
```

WHICH DECLARES EACH OF THE VARIABLES IDENT0 THROUGH IDENTK TO BE OF TYPE <TYPE>. THE <TYPE> DESIGNATION MAY BE FOLLOWED BY AN OPTIONAL CLAUSE

```
CHECK F
CHECK S
CHECK F,S
```

WHERE F,S REFER TO FETCHES AND STORES OF THE VARIABLE.
WHEN SUCH A SPECIFICATION IS MADE FOR A PARTICULAR
VARIABLE, THE OUTPUT LISTING INCLUDES A LIST OF ALL STATE-
MENTS WHICH REFERENCE OR ASSIGN TO THAT VARIABLE. (IN ADDITION,
A GLOBAL OPTION EXISTS TO CROSS-REFERENCE FETCHES AND STORES OF
ALL DATA ITEMS REGARDLESS OF THE CHECK SPECIFICATION.)

<TYPE> MAY BE A DEFINED TYPE, SIGNIFIED BY <IDENTIFIER>,
OR ONE OF THE SIMPLE TYPES INT, SINT, OR INDEX, OR
A COMPOUND TYPE STRING, ARRAY, OR POWERSET.

IF ANY FLAGS ARE TO BE USED FOR COMMUNICATION BETWEEN COOPERATI
PROCESSES, THEY ARE DECLARED IN THE FLAG SECTION OF THE
APPROPRIATE BLOCK.

```
FLAG
 <IDENT> (* CHECK CLAUSE *)
 <IDENT> (* CHECK CLAUSE *)
      .

END FLAG
```

(SEE SECTION 10.F, ASYNCHRONOUS CONTROL.) IN THE CASE OF FLAGS
THE CHECK OPTIONS ARE F,S,L,U WHERE L & U REFER TO LOCK AND
UNLOCK, EITHER EXPLICIT, OR IMPLICIT VIA WAIT, SET AND FLAGVAL.

7. EXPRESSIONS

ONE OF THE PRINCIPAL COMPONENTS OF THE ACTION SECTION OF A
PROGRAM IS THE EVALUATION OF EXPRESSIONS. IN THE SINGLE LANGUA
EXPRESSIONS ARE OF SIX TYPES, OF WHICH THREE ARE EXTREMELY
LIMITED:

A. ARRAY & RECORD TYPE EXPRESSIONS MAY CONSIST ONLY OF THE NAM
OF A DATA ITEM OF THAT TYPE. SUCH AN EXPRESSION IS USEFUL
ONLY FOR COPYING THE VALUES OF THAT ITEM VIA AN ASSIGNMENT
STATEMENT, OR FOR PASSING ARGUMENTS TO A PROGRAM BLOCK.

B. SCALAR TYPE EXPRESSIONS CONSIST OF THE NAME OF A SCALAR
TYPE DATA ITEM, OR OF THE NAME OF A VALUE OF A SCALAR
TYPE DATA ITEM. IN ADDITION TO COPYING AND ARGUMENT
PASSING, SUCH EXPRESSIONS ARE USEFUL IN EXPRESSING CONDITIO

C. POWERSET EXPRESSIONS ARE FORMED IN ACCORDANCE WITH THE

STANDARD NOTATION OF SET THEORY, USING THE OPERATOR +, *, -, FOR UNION, INTERSECTION AND RELATIVE COMPLEMENT, PLUS PARENTHESES, THE SYMBOLIC VALUES 'EMPTY' & 'ALL', AND NAMES OF DATA ITEMS WHICH ARE POWERSETS OF THE SAME SCALAR TYPE. IN SUCH AN EXPRESSION, THE NAME OF A VALUE OF THE BASE SCALAR TYPE IS INTERPRETED AS MEANING THE SET CONSISTING OF THAT ONE VALUE, EXAMPLE:

```
TYPE
  .
  .
 DIRECT = (NORTH,SOUTH,EAST,WEST)
  .
END TYPE
  .
VARIABLE
  .
 DREC,D            :POWERSET OF DIRECT
  .
END VARIABLE
  .
  .
  .
 D:=DREC+EAST
```

D. INT, SINT, OR INDEX TYPE EXPRESSIONS ARE FORMED IN THE CONVENTIONAL MANNER USING THE OPERATORS +,-,*,/,\, (THE LAST TWO SIGNIFYING QUOTIENT AND REMAINDER RESPECTIVELY) PLUS PARENTHESES, NAMES OF INT, SINT, OR INDEX TYPE DATA ITEMS AND FUNCTIONS. THE NUMERIC TYPE FUNCTIONS ARE

MAX (A,B)	EXPRESSIONS A AND B OF TYPE NUMERIC
MIN (A,B)	" " " " " " "
ABS(A)	THE TYPE OF EXPRESSION A MUST BE NUMERIC.
CONVSI(A)	THE TYPE OF EXPRESSION A MUST BE STRING.
FLAGVAL(FLAGID)	

NOTE THAT THE TYPE OF MAX OR MIN IS SINT IF EITHER OF ITS ARGUMENTS ARE OF TYPE SINT.

THESE FUNCTIONS GIVE RESPECTIVELY THE LARGER OF THE TWO VALUES, THE SMALLER, THE ABSOLUTE VALUE OF THE ARGUMENT, THE INTEGER REPRESENTED BY A STRING CONSISTING OF DECIMAL DIGIT CHARACTERS, AND THE VALUE OF THE DESIGNATED FLAG. IN ADDITION, ANY CHARMAP'S ARE INT TYPE FUNCTIONS.

E. STRING TYPE EXPRESSIONS ARE FORMED USING STRING REFERENCES, THE STRING TYPE FUNCTION CONVIS(A), PARENTHESES, AND THE STRING OPERATORS '+','-','/', AND '\'.

A STRING REFERENCE IS EITHER THE NAME OF A STRING TYPE DATA ITEM OR A SUBSTRING REFERENCE, WHICH CONSISTS OF THE NAME OF A STRING ITEM FOLLOWED BY ONE OR TWO INT TYPE EXPRESSIONS SEPARATED BY A COMMA AND ENCLOSED IN PARENTHESES. FOR EXAMPLE IF SAM IS DECLARED OF TYPE STRING, THEN

```
SAM
SAM(A+B)
SAM(C,D-E)
```

ARE EACH STRING REFERENCES. THE MEANING OF THE INT TYPE EXPRESSIONS ARE RESPECTIVELY A CHARACTER POSITION WITHIN THE STRING (STARTING FROM ZERO AT THE LEFT) AND THE LENGTH OF A SUBSTRING STARTING AT THAT POSITION. IF ONLY THE FIRST IS PRESENT, THE IMPLIED VALUE OF THE SECOND EXPRESSION IS 1.

THE MEANING OF THE OPERATORS IN STRING EXPRESSIONS IS AS FOLLOWS:

+ : CONCATENATION. A+B GENERATES A STRING CONSISTING OF THE CHARACTERS OF A FOLLOWED BY THE CHARACTERS OF B.

- : EXCISION. A-B GENERATES A STRING BY REMOVING FROM A THE LEFT-MOST OCCURRENCE OF THE CHARACTERS OF B.

/ : HEADER. A/B CONSISTS OF THOSE CHARACTERS OF A WHICH PRECEDE THE LEFT-MOST OCCURRENCE OF THE CHARACTERS OF B.

\ : TRAILER. A\B CONSISTS OF THOSE CHARACTERS OF A WHICH FOLLO THE LEFT-MOST OCCURRENCE OF THE CHARACTERS OF B.

THE FUNCTION CONVIS(A) GENERATES A STRING OF DECIMAL CHARACTERS REPRESENTING THE VALUE OF THE INT TYPE EXPRESSION A.

8. ASSIGNMENTS

ASSIGNMENTS ARE PROGRAMMED USING THE REPLACE, INCREMENT AND DECREMENT ACTIONS, WHICH RESPECTIVELY HAVE THE FORMS

```
        <REF LIST>:=<EXPRESSION>
        INCR <REF LIST>(*BY <EXPRESSION>*)
        DECR <REF LIST>(*BY <EXPRESSION>*)

    THE <REF LIST> IS EITHER THE NAME OF A DATA ITEM, OR A LIST
    OF SUCH NAMES, SEPARATED BY COMMAS.  IN THE LATTER CASE, THE
    EFFECT IS THE SAME AS IF A SERIES OF STATEMENTS WERE MADE,
    EACH ASSIGNING THE SAME VALUE TO SUCCESSIVE DATA ITEMS IN THE
    LIST.
    THE EFFECT OF THE REPLACE STATEMENT IS THAT THE <EXPRESSION> IS
    EVALUATED AND THE DATA ITEM TAKES ON THE RESULTING VALUE.
    THE INCREMENT AND DECREMENT STATEMENTS APPLY TO NUMERIC ITEMS
    ONLY, AND HERE THE DATA ITEM HAS THE VALUE OF THE <EXPRESSION>
    ADDED TO OR SUBTRACTED FROM THE PREVIOUS VALUE OF THE ITEM.
    IF THE <EXPRESSION> IS NOT PRESENT A VALUE OF 1 IS IMPLIED.

    THESE ARE THE MAIN ACTIONS WHICH ASSIGN VALUES TO DATA ITEMS
    AT RUN TIME.  ALL SUCH ASSIGNMENTS ARE CHECKED AT COMPILE TIME
    FOR COMPATIBILITY OF TYPE, THAT IS, THE TYPE OF THE RECEIVING
    ITEM MUST BE COMPATIBLE WITH THAT OF THE SOURCE ITEM, OR A
    WARNING OR ERROR MESSAGE RESULTS.  COMPATIBLE TYPES FOR A GIVEN
    RECEIVING ITEM TYPE ARE GIVEN BELOW.  IN THIS TABULATION, N
    AND M STAND FOR VALUES OF INSTANCES OF <NVAL>, WITH M<=N.

    RECEIVING TYPE          COMPATIBLE TYPES

      INT[N]                INT[M], INDEX=>(ARRAY[M])

      INDEX=>(ARRAY[N])     INT[M], INDEX=>(ARRAY[M])

      SINT[N]               INT[M], SINT[M], INDEX=>(ARRAY[M])

      STRING[N]             STRING[M]  (IN THIS INSTANCE, IF M>N AT
                                  RUN TIME THE RESULT IS DEFINED:
                            THE RECEIVING STRING IS FILLED TO ITS
                            CAPACITY WITH CHARACTERS OF THE SOURCE
                            STRING STARTING FROM THE LEFT-MOST
                            POSITION.  IF M < N, THE RESULT IS AS IF
                            THE SOURCE STRING WERE EXTENDED ON THE
                            RIGHT BY N-M BLANK CHARACTERS).

      SCALAR W              SCALAR w

      POWERSET OF X         POWERSET OF X, SCALAR X  (IN THE LATTER
                                  CASE, THE SCALAR IS CONVERTED TO
                                  A POWERSET CONSISTING OF THAT
                                  ONE SCALAR VALUE)
```

ARRAY[N] OF Y ARRAY[N] OF Z (TYPE OF Z COMPATIBLE WITH THAT OF Y).

R (A RECORD TYPE) R

IF THE TYPES ARE INCOMPATIBLE ONLY IN THAT M>N, THE COMPILER GENERATES A WARNING MESSAGE, AND ASSUMES THAT THE PROGRAM CAN GUARANTEE THAT THE SOURCE ITEM IS IN FACT WITHIN THE RANGE OF THE RECEIVING ITEM. IF THE SOURCE ITEM IS NOT WITHIN THE RANGE OF THE RECEIVING ITEM AT RUN TIME, RESULTS ARE UNDEFINED EXCEPT IN THE CASE OF STRING ITEMS NOTED ABOVE.

9. CONDITIONS

CONDITIONS ARE USED TO SELECT ALTERNATIVE ACTIONS TO BE PERFORMED BASED ON RELATIONSHIPS AMONG DATA ITEMS. THE BASIC COMPONENTS OF CONDITIONS ARE RELATIONS, AND A GENERAL CONDITION IS A BOOLEAN EXPRESSON FORMED FROM RELATIONS, PARENTHESES AND THE OPERATORS 'NOT', 'AND', AND 'OR'.

RELATIONS ARE FORMED BY A PAIR OF EXPRESSIONS OF SIMILAR TYPE SEPARATED BY A RELATIONAL OPERATOR. THE OPERATORS AND THEIR MEANING FOR THE VARIOUS TYPES ARE SUMMARIZED AS FOLLOWS:

TWO EXPRESSIONS OF TYPE	OPERATORS	MEANING
NUMERIC	=, ¬=, <=, >=, <, >	EQUAL, NOT EQUAL, LESS OR EQUAL, GREATER OR EQUAL LESS, GREATER
STRING	SAME	SAME
POWERSET	SAME	EQUAL, NOT EQUAL, CONTAINED, CONTAIN PROPERLY CONTAINED PROPERLY CONTAINS
SCALAR	=, ¬=	EQUAL, NOT EQUAL

IN THE CASE OF STRING RELATIONS THE NORMAL SORTING CONVENTIONS APPLY. THE SHORTER OF THE TWO STRINGS IS CONCEPTUALLY LENGTHENED TO MATCH THE LARGER BY ADDING BLANK

CHARACTERS ON THE RIGHT AND THE RELATIONS LESS AND GREATER ARE DETERMINED BY THE NORMAL ASCII COLLATING SEQUENCE VIA A LEFT TO RIGHT SCAN FOR FIRST NON-EQUAL CHARACTERS.

10. CONTROL STATEMENTS

A. GROUPING STATEMENTS

THERE ARE TWO CONSTRUCTS IN SINGLE WHICH HAVE THE EFFECT OF GROUPING A SEQUENCE OF STATEMENTS SO THAT THE SEQUENCE MAY APPEAR WHEREVER AN INDIVIDUAL STATEMENT IS CALLED FOR IN THE SYNTAX. THE FIRST OF THESE IS THE CONSTRUCT

```
BEGIN (*<GROUPID>*):
 <STATEMENT>
 <STATEMENT>
      .
      .
      .
END (*<GROUPID>*):
```

THE SECOND CONSTRUCT HAS THE SAME EFFECT BUT ALSO INCLUDES REPETITION OR LOOPING THROUGH THE STATEMENTS (SEE 10.C, LOOPING STATEMENTS).

```
CYCLE (*<GROUPID>*)(*<REPEAT CONTROL>*):
 <STATEMENT>
 <STATEMENT>
      .
      .
      .
NEXT (*<GROUPID>*):
```

THE OPTIONAL GROUPID MUST BE THE SAME IN BEGIN AND END, OR IN CYCLE AND NEXT, OR A WARNING MESSAGE IS GENERATED.

B. CONDITIONAL STATEMENTS

THERE ARE TWO TYPES OF CONDITIONAL STATEMENTS. THE FIRST IS OF THE FORM

```
IF <CONDITION> THEN
 <STATEMENT>
ELSE
```

<STATEMENT>

WHICH CAUSES THE FIRST OF THE TWO STATEMENTS TO BE EXECUTED IF THE CONDITION IS TRUE AND THE SECOND TO BE EXECUTED IF THE STATEMENT IS FALSE. A SECOND FORM,

```
IF <CONDITION> THEN
 <STATEMENT>
```

IS ALSO PERMITTED. THE AMBIGUITY OF A CONSTRUCT SUCH AS

```
IF <CONDITIONO> THEN
  IF <CONDITION1> THEN
     <STATEMENT>
  ELSE
     <STATEMENT>
```

IS RESOLVED BY ARBITRARILY FORCING THE ELSE TO BE ASSOCIATED WITH THE INNERMOST IF.

THE OTHER CONDITIONAL CONSTRUCT IS

```
CASE <NUMERIC EXPRESSION> OF
  0:<STATEMENT0>
  1:<STATEMENT1>
    .
    .
    .
  N:<STATEMENTN>
END CASE
```

HERE THE EXPRESSION IS EVALUATED AND THE RESULTING VALUE IS USED TO SELECT THE ONE STATEMENT WHICH IS TO BE EXECUTED. NO CHECKING IS DONE TO SEE IF THE VALUE LIES IN THE RANGE 0-N. THE NUMBERS PRECEDING THE COLON ARE FOR READABILITY ONLY. THE VALUES ARE DETERMINED BY POSITION IN THE LIST. IN BOTH THE IF AND CASE STATEMENTS SPECIAL TREATMENT IS GIVEN WHEN THE CONDITION OR NUMERIC EXPRESSION IS FULLY EVALUATABLE AT COMPILE TIME (INVOLVES ONLY CONSTANTS AND PARAMETERS). THE APPROPRIATE BRANCH IS SELECTED AT COMPILE TIME AND ONLY THE CODE FOR THAT BRANCH IS GENERATED.

C. LOOPING STATEMENTS

REPETITION OR LOOPING OF STATEMENT EXECUTION IS ACHIEVED BY VARIOUS FORMS OF THE CYCLE CONSTRUCT:

```
CYCLE (*<GROUPID>*)(*<REPEAT CONTROL>*):
 <STATEMENT>
 <STATEMENT>
      .
      .
      .
NEXT (*<GROUPID>*);
```

THE ABSENCE OF REPEAT CONTROL MEANS THAT THIS GROUP OF STATEMENTS IS TO BE EXECUTED REPEATEDLY UNTIL ONE OF THE STATEMENT CAUSES A TRANSFER OUT OF THE GROUP (SEE 10.D, TRANSFER STATEMENTS).

REPEAT CONTROL HAS TWO FORMS:

```
WHILE <CONDITION>

AND

FOR <IDENT>:=<RANGE0>,<RANGE1>,...,<RANGEN>
```

IN EACH CASE, WHEN A REPEAT OF THE GROUP IS CALLED FOR (BY A TRANSFER STATEMENT OR BY CONTROL REACHING THE ENDING 'NEXT' THE INDICATED TEST IS MADE, I.E., IS THE CONDITION FALSE OR HAS THE RANGE LIST BEEN EXHAUSTED. IF NOT, REPETITION BEGINS. IF YES, CONTROL PASSES TO THE FIRST STATEMENT PAST THE 'NEXT'.

RANGE IS THE FORM <NUMERIC EXPRESSION>, OR

(*<EXPRESSION0>*) TO <EXPRESSION1> (*STEP <EXPRESSION2>*)

THROUGHOUT AN EXECUTION UNDER FOR CONTROL THE VALUES OF THE EXPRESSIONS UPON FIRST ENTRY OF THE CYCLE STATEMENT ARE THOSE USED FOR REPETITION CONTROL AND TESTING, I.E., CHANGING THE VALUES OF THE EXPRESSIONS WITHIN THE SCOPE OF THE CYCLE DOES NOT AFFECT THE LOOPING CONTROL. THE IDENT SHOULD BE AN IDENTIFIER DECLARED TO BE A VARIABLE OF TYPE INT OR INDEX. UPON TERMINATION OF EXECUTING UNDER FOR CONTROL THE VALUE OF THIS VARIABLE IS THAT CORRESPONDING TO THE LAST REPETITION PLUS THE STEP. IF THE REPETITION WAS TERMINATED BY A LEAVE, THE VALUE OF THE CONTROL VARIABLE IS THAT WHICH IT HELD WHEN THE LEAVE WAS EXECUTED. THE CONTROL VARIABLE IS A NORMAL VARIABLE IN ALL RESPECTS, AND VALUES MAY BE ASSIGNED TO IT

WITHIN THE SCOPE OF THE 'CYCLE'.

DEFAULT VALUE OF NUMERIC EXPRESSION0 IS 0.
DEFAULT VALUE OF NUMERIC EXPRESSION2 IS 1.

D. TRANSFER STATEMENTS

TWO STATEMENTS PROVIDE FOR TRANSFERS OF CONTROL WITH RESPECT TO BEGIN OR CYCLE GROUPS:

```
LEAVE(*<GROUPID>*):

   AND

RECYCLE(*<GROUPID>*):
```

LEAVE REFERS TO THE NEAREST PRECEDING CYCLE OR BEGIN IF IT CARRIES NO GROUPID, OR THE GROUP NAMED OTHERWISE. (IF NO SUCH GROUP EXISTS A NON-FATAL ERROR RESULTS). THE EFFECT IS TO TRANSFER CONTROL TO THE FIRST STATEMENT PAST THE GROUP ENDING (END OR NEXT).

RECYCLE REFERS TO A PRECEDING CYCLE, AND TRANSFERS CONTROL TO WHATEVER TESTING AND INCREMENTATION IS CALLED FOR BY THE REPEAT CONTROL. THUS IT HAS THE SAME CONTROL FUNCTIONS AS NEXT, BUT DOES NOT TERMINATE THE STATEMENT GROUPING.

ON THE COMPILER SOURCE OUTPUT THE LEFTMOST TWO COLUMNS WILL SHOW A LEXICAL LEVEL FOR EACH STATEMENT. E.G.

```
10          IF ... THEN
11           <STAT>
10          ELSE
11           BEGIN:
12            <STAT>
 :              .
12            <STAT>
11           END:
10          <STAT>
 .
 .
 .
```

THE APPEARANCE OF A LEAVE STATEMENT WILL CAUSE A NOTATION

'FNN' TO APPEAR BESIDE THE LEVEL NUMBER, INDICATING THE LEVEL NUMBER TO WHICH CONTROL IS ADVANCING. SIMILARLY, RECYCLE CAUSES 'BNN' TO APPEAR.
EXAMPLE:

```
3            CYCLE SAM WHILE ... :
4             <STAT>
                 .
                 .
4            CYCLE AL FOR ... :
5             <STAT>
                 .
                 .
5F3           LEAVE SAM
                 .
                 .
5B3           RECYCLE SAM
              .
              .
4B4           NEXT AL
              .
              .
              .
3B3          NEXT SAM
3            <STAT>
```

THE NOTATION DIRECTS THE READER FORWARD OR BACKWARD TO THE FIRST OCCURRENCE OF THE INDICATED LEVEL NUMBER.

E. INTERPROGRAM CONTROL

SEVERAL STATEMENTS CAUSE TRANSFER OF CONTROL FROM ONE PROGRAM TO ANOTHER. THESE ARE:

```
CALL
TRANSFER  PROCEDURE (*(PARAMETER RECORD)*)
DEFER           TASK (*(PARAMETER RECORD)*)
START           TASK (*(PARAMETER RECORD)*)(*OR <STAT>*)
RETURN
STOP
```

CALL IS THE CONVENTIONAL PROCEDURE INVOCATION. THE CALLING PROGRAM GOES INTO WAIT STATUS AND WHEN THE CALLED PROCEDURE EXECUTES A RETURN STATEMENT CONTROL RETURNS TO THE CALLING

PROGRAM AT THE STATEMENT FOLLOWING THE CALL STATEMENT.

A CALLED PROCEDURE NEED NOT EXECUTE A RETURN STATEMENT BUT MAY INSTEAD EXECUTE A TRANSFER STATEMENT INVOKING ANOTHER PROCEDURE. THIS TERMINATES THE PROCEDURE EXECUTING THE TRANSFER AND MAKES THE PROCEDURE TRANSFERRED TO ACT LIKE THE CALLEE OF THE ORIGINAL CALLER, I.E., IF IT SHOULD EXECUTE A RETURN, CONTROL REVERTS TO THE ORIGINAL CALLER. IN TURN, THIS PROCEDURE MAY TRANSFER TO YET ANOTHER, ETC. UNTIL FINALLY SOME PROCEDURE IN THE CHAIN EXECUTES A RETURN.

THUS A PROCEDURE IS INVOKED BY A CALL OR A TRANSFER AND TERMINATES ITS EXECUTION BY A RETURN OR A TRANSFER.

A TASK IS A PROGRAM WHICH EXECUTES INDEPENDENTLY OF THE PROGRAM WHICH INVOKED IT, I.E., THE INVOKING PROGRAM DOES NOT GO INTO WAIT STATUS BUT EITHER TERMINATES (VIA THE DEFER INVOCATION) OR CONTINUES ASYNCHRONOUSLY (VIA THE START INVOCATION). NOTE THAT SINCE A PROCEDURE MUST TERMINATE BY EITHER A RETURN OR TRANSFER STATEMENT, IT MAY NOT EXECUTE A DEFER. ON THE OTHER HAND A TASK MAY NOT EXECUTE A RETURN OR TRANSFER SINCE IT HAS NO RELATIONSHIP TO A CALLER. A TASK TERMINATES BY EXECUTING EITHER A DEFER OR A STOP.

NOTE THAT EITHER A TASK OR A PROCEDURE MAY EXECUTE A CALL TO A PROCEDURE, IMPLYING WAIT STATUS FOR THE CALLER, OR A TASK, IN WHICH CASE THE STARTER CONTINUES ASYNCHRONOUSLY.

IN SUMMARY THE INVOKING OF ONE PROGRAM BY ANOTHER AND THE DISPOSITION OF THE INVOKER, IS AS FOLLOWS

		INVOKEE	
INVOKER	DISPOSITION	TASK	PROCEDURE
TASK	WAIT	*	CALL
	CONTINUE	START	-
	TERMINATE	DEFER	-
PROCEDURE	WAIT	*	CALL
	CONTINUE	START	-
	TERMINATE	-	TRANSFER

- MEANS THAT THE COMBINATION IS NOT ALLOWED.

* MEANS THAT THE EFFECT OF INVOKING A TASK AND WAITING FOR ITS COMPLETION OR FOR SOME OTHER EVENT CAN BE ACHIEVED BY USING THE START AND THEN EXECUTING A WAIT FOR THE APPROPRIATE

EVENT (SEE 10.F, ASYNCHRONOUS CONTROL STATEMENTS.)

THE START STATEMENT INCLUDES AN 'OR' OPTION FOR USE IN THE CASE WHERE THE TASK TO BE INVOKED IS CURRENTLY NOT FREE TO BE INVOKED. IF THE 'OR' OPTION IS OMITTED, THE INVOKER GOES INTO WAIT STATUS UNTIL THE INVOKEE IS FREE, AT WHICH TIME THE INVOKE AND INVOKEE ARE SCHEDULED FOR RESUMPTION (AT THE STATEMENT AFTER THE COMPLETE START STATEMENT) AND INITIATION, RESPECTIVELY. IF THE 'OR' OPTION IS USED, AND THE INVOKEE IS NOT BUSY, INVOKER AND INVOKEE ARE IMMEDIATELY SCHEDULED FOR RESUMPTION AND INITIATION RESPECTIVELY. IF THE INVOKEE IS BUSY, CONTROL IS RETURNED TO THE STATEMENT WITHIN THE START AND FOLLOWING THE 'OR'. NO RECORD IS KEPT OF THE ATTEMPTED INVOCATION. THE INVOKER HAS SIMPLY FAILED TO GET THE TASK STARTED, AND MAY TRY AGAIN AT SOME LATER TIME.

F. ASYNCHRONOUS CONTROL

SYNCHRONIZATION AND CONTROL AMONG ASYNCHRONOUSLY EXECUTING PROGRAMS MAY BE ACHIEVED BY THE USE OF FLAGS AND EVENTS. THE FLAG IS AN IDENTIFIER <FLAGID> DECLARED IN THE FLAG SECTION. ITS TYPE IS INT[99], AND ITS VALUE IS PRESET TO ZERO. AN EVENT IS A BOOLEAN COMBINATION, USING 'OR' AND 'AND' OPERATORS (NO 'NOT' OPERATORS) OF SIMPLE EVENTS, WHICH HAVE THE FORM

FINISH <PROGID>

OR

<FLAGID> <RELOP><EXPRESSION>

WHERE <RELOP> IS ONE OF THE NUMERIC RELATIONAL OPERATORS (SEE 9. CONDITIONS) AND <EXPRESSION> IS OF NUMERIC TYPE (INT OR INDEX). FINISH <PROGID> OCCURS WHEN THE PROGRAM NAMED EXECUTES ONE OF STOP, TRANSFER, OR DEFER. THE ACTION STATEMENTS INVOLVED ARE

WAIT <EVENT>

SET <FLAGID> TO <EXPRESSION>

UNLOCK <FLAGID>

LOCK <FLAGID> (* OR <STAT> *)

IN ADDITION, THERE IS A FUNCTION FLAGVAL (<FLAGID>) WHICH MAY BE USED IN EXPRESSIONS.

A FLAG CAN BE REFERENCED ONLY AFTER THAT PROGRAM HAS SUCCESSFULLY EXECUTED A LOCK OF THE FLAG. ONCE THE LOCK IS EXECUTED, CONTROL IS RETURNED TO THE STATEMENT AFTER THE COMPLETE LOCK STATEMENT. IF THE FLAG HAS CURRENTLY BEEN LOCKED BY ANOTHER PROGRAM, CONTROL EITHER REVERTS TO THE STATEMENT AFTER THE 'OR', OR WAITS TILL THE FLAG IS UNLOCKED IF NO 'OR' OPTION IS PRESENT.

A FLAG VALUE IS REFERENCED EITHER BY AN EVENT OR BY THE FLAGVAL FUNCTION. IN EITHER TYPE OF REFERENCE A CHECK IS MADE TO SEE THAT THE FLAG HAS BEEN LOCKED BY THIS PROGRAM. IF NOT, A RUN-TIME ABORT OCCURS. IF THE REFERENCE IS BY WAIT <EVENT> THE IN THE EVENT ARE TESTED TO SEE IF THE EVENT HAS OCCURRED. IF NOT, THEY ARE UNLOCKED AND THE PROGRAM GOES INTO WAIT STATUS FROM THEN ON, WHENEVER A FLAG VALUE CHANGES ALL EVENTS REFERENCING THAT FLAG WILL BE TESTED. FOR THOSE EVENTS WHICH HAVE OCCURRED, THE PROGRAMS IN WAIT FOR THAT EVENT WILL BE SCHEDULED FOR RESUMPTION AT THE STATEMENT FOLLOWING THE WAIT.

THE SET STATEMENT CAUSES AN ABORT IF THE FLAG HAS NOT BEEN LOCKED BY THE PROGRAM. THE TYPE RULES FOR EXPRESSION ARE THOSE OF ASSIGNMENT TO A VARIABLE OF TYPE INT[99]. THE FINAL ACTION OF THE SET IS TO UNLOCK THE FLAG. IF A FLAG HAS BEEN REFERENCED ONLY BY LOCK AND FLAGVAL, AND NOT CHANGED BY SET, THEN THE PROGRAM MUST UNLOCK THAT FLAG BEFORE TERMINATING.

11. ARGUMENTS FOR PROGRAMS

AS HAS BEEN SEEN IN SECTION 5, DEFINITIONS, THE LINK DEFINITION OF THE PARENT BLOCK DESCRIBE THE STRUCTURE OF THE ARGUMENTS OF A PROGRAM. THIS IS UNLIKE THE CONVENTIONAL HIGH LEVEL LANGUAGE WHERE THE PROGRAM DEFINES THE TYPE OF ITS ARGUMENTS. THE PRESENT ARRANGEMENT ALLOWS THE COMPILER TO COMPILE AND TYPE CHECK INVOCATIONS OF THE PROGRAM INDEPENDENT OF THE PROGRAM BLOCK ITSELF, AND ALSO FACILITATES OVERLAY STRUCTURES.

ANYWHERE WITHIN THE SCOPE OF THE BLOCK WHICH HAS DEFINED A LINK TO PROGRAM IDENT WE MAY HAVE

```
    .
    .
    .
    VARIABLE
      .
```

```
        .
        .
        B           : IDENT
        .
    END VARIABLE
        .
  / CALL     \
 /  DEFER    \
 \  TRANSFER /    IDENT (B)
  \ START    /
```

IT SHOULD BE RECALLED THAT THE DEFINITION OF PROGRAM IDENT ALSO CREATES A RECORD TYPE DEFINITION CALLED IDENT, AND HENCE NAMING A RECORD OF THAT TYPE SERVES TO SPECIFY ALL THE PARAMETERS THAT PROGRAM REQUIRES.

WITHIN A PROGRAM, REFERENCES TO ITS OWN INVOCATION PARAMETERS ARE MADE BY CITING THE FIELD NAME QUALIFIED BY THE SYMBOL '@'. EXAMPLE:

```
    .
    .
    TYPE
        .
     SAM = TASK LINK
       A      :INT[£Q]
       B      :STRING[£P]
     ASSIGNS
       C      :ARRAY[£Q] OF INDEX->C
     END SAM
        .
    END TYPE
        .
        .
    VARIABLE
        .
        .
        D           :SAM
        .
    END VARIABLE
        .
        .
     START SAM(D)
        .
        .
 TASK SAM
        .
        .
```

```
        TEMP:=@.B

    THIS LAST STATEMENT HAS THE EFFECT OF REFERENCING THE 2ND
    COMPONENT OF THE RECORD D WHICH WAS USED AS THE ARGUMENT IN
    THIS PARTICULAR INVOCATION OF SAM.

    THE COMPILER WILL ISSUE A DIAGNOSTIC IF A PROGRAM ASSIGNS A
    VALUE TO ANY OF ITS ARGUMENTS NOT LISTED IN AN ASSIGNS OPTION
    OF THE LINK DEFINITION FOR THE PROGRAM.



12. MACHINE SPECIFIC STATEMENTS

    IN ORDER TO FACILITATE WRITING OF DEVICE DEPENDENT PROGRAMS
    SUCH AS DEVICE DRIVERS AND DEVICE DIAGNOSTICS, CERTAIN MACHINE
    DEPENDENT STATEMENTS ARE ALLOWED. THE USE OF THESE STATEMENTS
    IS CONTROLLED BY PASSWORDS GIVEN IN COMPILER DIRECTIVES. THE
    APPEARANCE OF A MACHINE-SPECIFIC CONSTRUCT IN A PROGRAM WILL
    PRODUCE A FATAL DIAGNOSTIC UNLESS

        • THE COMPILATION PASSWORD AUTHORIZES USE OF THAT
          CONSTRUCT, AND

        • THE CONSTRUCT BELONGS TO THE MACHINE FOR WHICH
          OBJECT CODE IS BEING GENERATED.

    A.  S/10 PROCESSOR 20, 21 AND S/11 DPU:

        THREE CONSTRUCTS ARE GIVEN:

        • INP A,B,C (* ,D *)
          AND
        • OUT A,B,C (* ,D *)

          WHERE

          A IS A STRING REFERENCE (SOURCE OR DESTINATION FIELD)

          B, C ARE EXPRESSIONS EVALUATING TO INT [99] (DEVICE
          NUMBER & CONTROL OPTIONS)  NOTE: DISC I/O IS NOT
          SUPPORTED.

          D IS AN OPTIONAL STRING REFERENCE OF LENGTH 4 (ID)

        • STATUS A(* ,B *)
```

WHERE

A IS A VARIABLE OF TYPE INT [6] (CONDITION CODE)

B IS AN OPTIONAL STRING REFERENCE OF LENGTH 4 (ID)

B. S/11 MPU

TWO CONSTRUCTS CONCERN I/O:

- INP A,B,C
- OUT A,B,C

WHERE

A IS A STRING REFERENCE OF LENGTH 1

B, C ARE AS FOR THE DPU

THREE ADDITIONAL CONSTRUCTS CONCERN INTERRUPT CONTROL:

- ENABLE A
- DISABLE A
- RETURN ENABLE A

WHERE A IS AN EXPRESSION EVALUATING TO INT[16].

13. BLOCK TERMINATION

THE END OF EACH COMPILATION BLOCK (CONTEXT, DATA OR PROGRAM) IS SIGNIFIED TO THE COMPILER BY THE SPECIAL SYMBOL COMBINATION

!

(UNDERLINE, EXCLAMATION, UNDERLINE), EXCEPT THAT THE LAST BLOCK OF THE CURRENT COMPILATION RUN IS SIGNIFIED BY THE OS END-OF-FILE CONVENTION

/*

EACH OF THESE MUST BEGIN IN COLUMN 1 OF THE INPUT LINE IN ORDER TO BE RECOGNIZED.

14. PROGRAMMING CONSIDERATIONS

A. INPUT/OUTPUT. MOST I/O, INCLUDING ALL DISC I/O, WILL BE DONE BY INVOKING ASSEMBLY LANGUAGE SUBPROGRAMS. THE MACHIN SPECIFIC STATEMENTS ARE MAINLY FOR DIAGNOSTIC WORK AND SHOU NOT BE USED FOR NON-DEVICE DEPENDENT LOGIC SUCH AS OPERATOR COMMUNICATION, SETTING AND READING TIMING CLOCKS, ETC. A SIMPLE ROUTINE FOR ACCEPT AND DISPLAY-TYPE OPERATIONS WILL BE PROVIDED.

B. OTHER MACHINE CONSIDERATIONS. CERTAIN ELEMENTS OF THE LANGUAGE WILL PRODUCE MARKEDLY LESS EFFICIENT CODE ON ONE PROCESSOR THAN ON ANOTHER. THE MOST OBVIOUS EXAMPLE OF THIS IS POWERSET VARIABLES, WHOSE NATURAL REPRESENTATION IS A BIT STRING. ON THE DECIMAL PROCESSORS NOT HAVING A LOGIC INSTRUCTION THE REPRESENTATION WILL BE ONE CHARACTER PER BIT.

C. MACHINE DEPENDENCIES. IN ADDITION TO THE MACHINE-SPECIFIC OPERATIONS OF SECTION 12, CERTAIN OTHER MACHINE DEPENDENCIE ARE POSSIBLE IN PROGRAMS WRITTEN USING SINGLE. THEY INCLUD

- LINKING OF HAND-CODED PROGRAM BLOCKS.
- CREATION OF OVERLAY STRUCTURES (DEPENDENT ON PARTITION AND/OR MEMORY SIZE).
- DEVICE-DEPENDENT LOGIC.
- USE OF 7-AND 8-BIT CHARACTER SETS.
- ANY USE OF OPERATIONS WHICH OVERFLOW: VARIABLE SIZE, SUBSCRIPT BOUNDS, ETC., SINCE THESE ARE UNCHECKED AT RUN-TIME.
- EXCEEDING THE CAPACITY OF THE TABLES IN THE SMALLEST LINKER UTILITY.

D. DEBUG. SOME FORM OF SYMBOLIC DEBUGGING CAPABILITY IS CLEARLY DESIRABLE. NO PLANS HAVE YET BEEN MADE FOR THE FORM THIS SHOULD TAKE. READERS OF THIS DOCUMENT ARE

INVITED TO PROPOSE SPECIFIC FEATURES OF A DEBUG PACKAGE.