FASBOL -- 6000

A SNOBOL4 COMPILER FOR THE CDC 6000 SERIES

by

I. Richard Strauss, Jr.

Memorandum No. ERL-M401

August 1973

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

FASBOL -- 6000

A SNOBOL4 COMPILER FOR THE CDC 6000 SERIES

by

I. Richard Strauss, Jr.

August, 1973

Submitted in partial fulfillment of the
requirements for the Master of Science
in Electrical Engineering and Computer Sciences
University of California, Berkeley, California

ACKNOWLEDGEMENT

The FASBOL - 6000 compiler described herein is the culmination of a year and six month's work during which time I received assistance from several sources.

Professor W. D. Maurer suggested the project topic and provided guidance during the entire period. I received financial assistance as his Research Assistant under NSF Grant GJ-31612.

I am indebted to Dr. Paul Santos who designed and implemented the first FASBOL compiler, on which the design of FASBOL - 6000 is largely based. He also helped me to understand the internal workings of his compiler and gave me suggestions for my own.

Paul McJones of the University Computer Center staff, one of the implementors of CAL SNOBOL, assisted me in comprehending the implementation of CAL SNOBOL and offered suggestions based on practical experience.

Gene Dronek, also of the University Computer Center staff, provided guidance through the maze of the SCOPE operating system and with COMPASS, version 2.

And finally I would like to thank the CDC 6400 (machine A) at the University Computer Center with whom I spent many a long night.

TABLE OF CONTENTS

CHAPTER 1

Introduction

The SNOBOL programming language (1) was developed and first implemented at the Bell Telephone Laboratories, Inc. in New Jersey. It was intially conceived as a string-manipulation language. The initial implementation was on an IBM 7094. The language was further developed at Bell Labs and evolved into SNOBOL3 (2,3) and SNOBOL4 (4,5). It grew to encompass not only string-manipulation but also many nonnumeric features not found in other programming languages. The implementors of SNOBOL4 developed a reasonably machine independent set of macros for SNOBOL4 compilation which were consequently implemented on many different computers. Today SNOBOL4 is a widely used and disseminated language (6). Some of the machines on which it has been implemented are the IBM System/360, UNIVAC 1108, GE 635, CDC 3600, CDC 6000 series, PDP-10, SIGMA 5/6/7/8/9, XDS 940, RCA SPECTRA 70, and ATLAS 2.

Until recently all SNOBOL4 implementations shared two features: they were interpreters rather than compilers and their speed of execution of source programs was very slow compared to other programming languages such as FORTRAN or COBOL. In 1969 a proprietary package for the IBM S/360 called SPITBOL (7) was released. It appears to be an incore system which compiles much of the SNOBOL source program as an absolute program and then immediately executes it. Complex pattern matches and string concatenations are apparently executed interpretively. Details of the internal workings of the SPITBOL compiler are unavailable. It is known to be ten to twenty times faster than the Bell Labs interpreter, a significant increase.

In 1969 Paul J. Santos, Jr. began work on a pure compiler for SNOBOL4. This system, which is known as FASBOL, compiles all features of the SNOBOL4 language including patterns. The resulting object program can then be executed. Subprograms written in FASBOL, FORTRAN or assembly language can be linked with FASBOL main programs. In addition, object programs can be segmented into overlay structures. The first FASBOL compiler (8) was written for the UNIVAC 1108 and was operational in October of 1971. It implemented SNOBOL4, version 2 (4). The second

FASBOL compiler, FASBOL II (9), was also written by Dr. Santos. It was implemented for the PDP-10 and was operational in August, 1972. It reflected the SNOBOL4 language of version 3 (5). These two systems proved conclusively that SNOBOL4 programs could be compiled with up to two orders of magnitude increase in execution speed over interpreters.

The first interpreter for the CDC 6000 series was implemented at the Institute for Defense Analyses at Princeton University from the Bell Labs macros (10). Its chief drawback was, of course, its slow execution speed. It also required a large amount of core (60-70K) for execution. In 1968 an interpreter for SNOBOL4 was developed at the Computer Center of the University of California, Berkeley by Charles Simonyi and Paul Mc Jones (11). This interpreter is known as CAL SNOBOL. CAL SNOBOL is written in COMPASS (the assembly language for the CDC 6000 series). The SNOBOL source program is translated into 60 bit micro instructions, each of which contains the address of the routine that executes the instruction. The micro code is then executed interpretively. It features very fast compilation speed (on the order of 15,000 lines per minute) and execution speed which is several times faster than the IDA implementation. It does not, however, implement all of SNOBOL4, version 2. It is well suited for its purpose which is to process a large number of student jobs at the University Computer Center.

It became apparent that there were some uses for which CAL SNOBOL was not well suited. Professor W. D. Maurer and some of his graduate students in the Department of Electrical Engineering and Computer Sciences at the Berkeley campus of the University of California were writing large SNOBOL4 programs in the areas of program verification and semantic descriptions of programming languages. These often required several minutes of CPU time to execute on the CDC 6400 using CAL SNOBOL. In addition, problems with the garbage collection scheme in CAL SNOBOL prevented some of them from running at all.

With this in mind, work was begun by the author in 1972 on an implementation of FASBOL for the CDC 6000 series based on the design of Dr. Santos. This system, which is now known as FASBOL - 6000, consists of a compiler written in FASBOL - 6000 of some 1200 source statements. The compiler accepts FASBOL - 6000 source programs and produces COMPASS assembly code. This code may then be assembled, linked with the FASBOL - 6000 runtime routines and executed. FASBOL - 6000 is a superset of SNOBOL4, version 2 and CAL SNOBOL with a few elements missing from each. A complete description of the language and how to use the compiler is given in chapter 2. The FASBOL - 6000 runtime system comprises approximately 100 subroutines written in COMPASS totalling 5000 source lines. A description of the internal workings of the compiler and runtime systems is given in chapter 3. The compiler was initially bootstrapped using CAL SNOBOL and is now capable of compiling itself. In fact it has now gone through four generations of self-compilation. As a test of consistency the current version was used to compile itself. The generated code was saved and then executed to once again compile the source of the compiler. The two object files were then compared. They were identical.

The compiler is currently being re-written to take advantage of certain SNOBOL4 features unavailable in CAL SNOBOL (principally the REPLACE function and the @ operator). This should afford a two to three times increase in compilation speed, but of course will not affect the execution speed. At the moment programs compiled by FASBOL - 6000 execute at approximately two or three times as fast as CAL SNOBOL. There are several optimizations known to the author which can increase this to ultimately four or five times as fast as CAL SNOBOL. These will be discussed in chapter 4.

CHAPTER 2

2. Language Description and Use

Most SNOBOL4 (version 2) and CAL SNOBOL programs will run with no changes under FASBOL - 6000. Familiarity with SNOBOL4 and CAL SNOBOL is assumed. Only language differences will be described here.

2.1.1 SNOBOL4 features not implemented

A complete list of unimplemented features is given in table 1. These are for the most part necessitated by the differences between a compiler and an interpreter. Their inclusion would result in a bulkier and less efficient runtime library.

2.1.2 SNOBOL4 features implemented differently

Unary . returns a value of type NAME, never a type STRING. Indirection (unary $) applied to a value of type NAME returns the same value. I.E. if

```
N = .A[10]
```

, then

```
$N = $N + 1
```

is equivalent to

```
A[10] = A[10] + 1
```

but executes faster.

The second argument to the primitive functions INPUT() and OUTPUT() is a string representing a file name rather than a number. The third argument to the primitive function OUTPUT() is a carriage control character rather than a FORTRAN FORMAT string. This prefix character (which may be the null string) is added to the front of each record on output for the associated variable. The optional fourth argument to OUTPUT() is the record length (if omitted 132 is assumed) in characters. The initial I/O association are:

```
OUTPUT(↑OUTPUT↑,↑OUTPUT↑,↑ ↑, 132)
INPUT(↑INPUT↑, ↑INPUT↑, 72)
```

The file name elements of these two associations can be overridden by control card specifications (see section 2.3).

TABLE 1

SNOBOL4 (version 2) Features Not Implemented

1. EVAL(), CODE() and direct GOTOs.
2. Datatypes CODE and EXPRESSION.
3. Non-literal protypes for DEFINE() and DATA().
4. All features, functions and keywords dealing with tracing except for ∧STNTRACE (see section 2.1.3).
5. Predefined VALUE() field.
6. Redefinition, OPSYN() or APPLY() of primitive functions, fields and pattern variables.
7. Two features of QUICKSCAN mode
   a. Continual comparision of the number of characters remaining in the subject string against the number of characters required.
   b. Assumption that unevaluated expressions must match at least one character. (This implies that left-recursive pattern definitions will loop forever.)
8. The keywords ∧ABORT, ∧ABEND, ∧ARB, ∧BAL, ∧DUMP, ∧FAIL, ∧FENCE, ∧REM, ∧SUCCEED.
9. The primitive functions CLEAR() and BACKSPACE().

The 6000 series character set requires the following changes:

360	6000
<,>	[,]
?	↱
@	↓
|	∨ or //
&	∧
:	: or /

2.1.3 Additions to SNOBOL4

The following compiler directives have been added:

-CODE, -NOCODE turn on and off the listing of object code between statements. The initial mode is off.

-SPACE N spaces N (or 1 if missing) lines in the source listing.

-NEWSTNO N resets the statement number to any value (N) greater than zero.

-EJECT causes a page eject in the source listing and object file.

The following primitive functions have been added:

(a) REALCH (CLASS) like BREAK() but ignores break characters inside substrings delimited by ≠ or ↑

(b) NSPAN(CLASS) equivalent to SPAN() ∨ NULL

(c) SUBSTR(STRING, INTEGER1, INTEGER2) equivalent to pattern match

```
STRING TAB(INTEGER1) LEN(INTEGER2) . VAL
```

but much faster and less space consuming.

(d) DUPL(STRING, INTEGER) returns the STRING formed by duplicating the STRING argument the INTEGER number of times.

(e) L PAD(STRING, LEN, PADCHR) returns the STRING formed by padding STRING on the left with PADCHR characters to a length of LEN. If STRING is already too long, it is returned unchanged; if PADCHR has more than one character only the first is used. If the third argument is null, blanks are used.

(f) RPAD(STRING, LEN, PADCHR) pads to the right.

(g) The following CAL SNOBOL primitive functions are implemented:

ENDGROUP()	ALPHABET()
EOI()	FNCLEVEL()
EORLEVEL()	MAXLNGTH()
ANCHOR()	STCOUNT()
IF()	STLIMIT()

(h) The ∧STNTRACE keyword when not equal to zero causes the current statement number to be printed before execution of the statement. ∧STNTRACE is initially zero.

2.2 Differences with CAL SNOBOL

The primitive functions TYPE(), NEXTVAR(), COMPILE(), and FREEZE() are unimplemented. Only [,] are allowed for array brackets; (/,/) are not allowed. ITEM() and PROTOTYPE() are implemented as in SNOBOL4. FASBOL - 6000 contains additions to CAL SNOBOL too numerous to mention but which can be inferred from sections 2.1.1, 2.1.2, 2.1.3 and (12).

2.3 Running a FASBOL - 6000 program

Assuming that the files FASBOL and FASLIB exist as COMMON files and have been so declared or exist as permanent files and have been ATTACHed the following control cards will compile and execute a FASBOL - 6000 program:

RFL,65000.   65K is the current minimum to compile. The fieldlength is automatically increased if needed. This figure should come down in future versions.

FLGO,FASBOL,INFIL,OUTFIL,BUFSIZ,HASHWID.

where INFIL is the name of the source file (INPUT is the default). OUTFIL is the name of the listing file (OUTPUT is the default). BUFSIZ is the size of I/O buffers in octal (should be a multiple of 100 octal + 1). 201 is the default. HASHWID is the width (in octal) of the hash code. The bucket table is then two to the power HASHWID words long. (The default is 6). For example,

FLGO,FASBOL,SOURCE,,1001.

will compile a program from the file SOURCE, with listing to the file OUTPUT. Buffer size is 1001 (octal) and hash code width is 6.

X,COMPASS,I=ASM,S=0,L=0.

Assembles the object program (which the compiler generates as the file ASM) using COMPASS, version 2.4 which is much faster than COMPASS, version 1.2. Omitting the L=0 option will provide a listing of the object code (at a price of increased assembly time).

CLDR,LIB=FASLIB,NOMAP,GO=INFILE,OUTFILE,BUFSIZ,HASHWID.

loads and executes the assembled program. The NOMAP option suppresses the load map. The GO parameters provide optional control of (a) the initial associations of the variables INPUT, OUTPUT, (b) the I/O buffer size and (c) the hash code width. The defaults are the same as for the FASBOL card. For example, if the parameters

GO=DATA,,,4.

are used, references to the variable INPUT will cause a record to be read from the file DATA and the hash bucket table will be 16 words long.

CHAPTER 3

3. The FASBOL - 6000 Compiler and Runtime System

As mentioned before, the design of the FASBOL - 6000 compiler is based on the design of Dr. Santos' original FASBOL compiler. The implementation design is significantly different, of course, since it runs on the CDC 6000 series under the SCOPE operating system rather than the UNIVAC 1108 under the EXEC II operating system. In addition, the design of several elements of the compiler and runtime is completely different. These are explained in sections 3.1 and 3.2.

3.1 The FASBOL - 6000 Compiler

The FASBOL - 6000 compiler is a FASBOL - 6000 program which accepts programs written in FASBOL - 6000 and produces COMPASS object code. Writing the compiler in FASBOL - 6000 proved a significant saving in effort. Approximately 90% of the coding effort involved the runtime library (which is written in COMPASS). Probably over 95% of the time spent debugging the system involved the runtime library. In addition coding the compiler in FASBOL - 6000 provided a mechanism to check out the library and demonstrates that FASBOL - 6000 is a useful language for compiler implementation.

The syntax of SNOBOL4 and hence of FASBOL - 6000 is such that it lends itself for the most part to ad hoc techniques for parsing. The exception is the parsing of expressions. Dr. Santos' FASBOL compiler uses a table-driven operator precedence scheme for such parsing. The FASBOL - 6000 parser is a straight-forward finite state machine with a single pushdown store which drives recursively defined semantic routines. The machine has 6 states, 3 input types, 5 stack entry types and 13 action routines.

The code generator emits code as it is generated. The FASBOL compiler generates a code tree which is later walked to produce the final code. As in FASBOL, patterns are compiled as two separate sections of code. One section is the code to evaluate the pattern parameters before pattern pattern matching is done. The other section is a re-entrant subroutine which corresponds to the pattern during the execution of a pattern match. The rest of FASBOL - 6000 is compiled as a call on an appropriate library routine.

3.2 The FASBOL - 6000 Runtime System

The FASBOL - 6000 runtime library consists of some 100 subroutines written in COMPASS. Only those routines which are actually needed during execution are linked with the generated code for the user's program. If the entire library were linked together it would total only 12k (octal) or 5k (decimal). As long as programs don't generate the need for additional storage they can execute in a relatively small field length.

The CDC 6000 series is not the ideal computer for running SNOBOL4 programs. The essential operations in SNOBOL4 are on strings of characters rather than numbers. The CDC machines have the justifiable reputation for being number crunchers. They are not character addressable as are the IBM S/360 machines. In fact the 6000 series have no character operations at all. Strings in FASBOL - 6000 are stored in blocks packed 10 characters to a word. To access a particular character it is necessary to get the proper word into a register, form a mask and do a mask operation, usually followed by a shift. Often compilers have internal tables with several fields packed into one word for space economy. It is not possible in one instruction to access a field which comprises a portion of one computer word. Once again a load, mask and shift or perhaps a load, shift, shift is required. Suppose there is an internal table where each word of the table consists of four 15 bit signed fields. Suppose further that one wants to put in register X6 the value of the second of these fields of the word which is in register X5, without destroying X5. The following code would be necessary:

```
BX6     X5
LX6     15
AX6     45
```

The fact that three instructions are required along with the necessity of remembering the values 15 and 45 lends itself to error, as well as making the code fairly undescipherable. It is for this reason that the author developed a series of macros which permit the one-line symbolic manipulation of partial words. This includes register to storage, storage to register, and register to register operations. These operations are used only in the runtime library, not in the generated code.

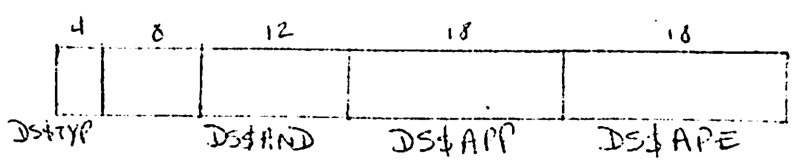
The basic element of this macro system is the FDEF (an acronym for field definition). The form of the FDEF is

```
NAME        FDEF        SBIT,LBITS,WORD,SIGN
```

where

SBIT	is the starting bit of the field (0 is the leftmost bit, 59 is the rightmost).
LBITS	is the length of the field in bits.
WORD	is the base word address of the field (if any).
SIGN	is 1 if the field is signed; 0 or missing if unsigned.

For example, the fields in the following word



are defined as.

```
DS$TYP      FDEF      0,4,0,1
DS$AND      FDEF      12,12,0,1
DS$APP      FDEF      24,18,0,1
DS$APE      FDEF      42,18,0,1
```

The FDEF of an entire word is 0,60,0,1.

There are seven operations, defined by the use of an elaborate macro structure, which operate on FDEF'd fields. They generate the optimal code for each operation. The macro DEFINE appears at the start of each routine in the library. This sets the values for X.MASK$,X.VOL$, and X.STOR$ which are used by some of the operations. The default values are X0, X5 and X7, respectively. The operations themselves are

(a) LD XR,FDEF+OFFSET

which loads a field into an X register, where OFFSET and the base address of the FDEF are combined in determining the actual address. OFFSET ↔ register ± constant or ± constant or zero or missing. OFFSET can also be BR±BR, XR+BR or AR±BR if the base address is zero.

(b) ST XR,FDEF+OFFSET

which stores the rightmost LBITS of the X register into the FDEF field of the addressed word. The original value of XR as well as the rest of the addressed word are preserved.

(c) SD XR,FDEF+OFFSET

which acts like ST except that the XR is destroyed (but not the rest of the addressed word.)

(d) GXR1 XR2,FDEF

which "gets" the field defined by FDEF from XR2 and right justifies it into XR1.

(e) SP XR,FDEF

which takes a field which is right justified in XR and "shifts it into place", i.e. shifts it left 60-SBITS-LBITS or generates no code if SBITS+LBITS = 60.

(f) TXR1 XR2,FDEF

which rotates XR2 to XR1 so that FDEF is right justified (works only for 18 bit fields - otherwise generates an error)

```
(h)  LJ       XR,FDEF
```

which left justifies the FDEF

This system has the additional benefit of allowing easily implemented changes to field definitions. To change the size or arrangement of fields it is merely necessary to change the appropriate FDEFs and re-assemble the routines! No searching through listings for references to non-symbolic fields. Note, too, that FDEF'd fields show up in the cross reference listing.

In FORTRAN a location is assigned at compile time to each variable in the program. A reference to the variable consists at runtime of a load of the value of the variable from the location. This is not possible in SNOBOL4 since the variables are not dedicated to a particular TYPE. So although a location is assigned at compile time to all natural variables in FASBOL - 6000, a reference consists at run-time of a load of [illegible] from the location of the current descriptor for the variable. This descriptor contains a field which gives its TYPE and (generally) a pointer to the current value. Integers, however have their value in the descriptor. Also strings of less than 8 characters have their value in the descriptor. Longer strings have a descriptor which points to the first word of the block where the value is packed ten characters per word. During pattern matches, the subject string is unpacked one character per word. Array descriptors contain pointers to the dope vectors as well as the element descriptor block. Diagrams for each descriptor format are given at the end of this section. The names of fields given in the diagrams are the FDEF names for the field.

The runtime routines have a consistent set of register conventions. B3,A3,X3,B4,A4, and X4 are dedicated to use by the pattern matching routines. Register B7 is used as a constant one throughout the library. Routines called from inline code may use any non-dedicated registers.

Second level routines (those called by the routines called by inline code) have a restricted set of registers they may use which varies with the routine. Input to primary routines uses the ES stack and register X1. The 1st through n - 1st arguments are stacked on ES. The last argument is in X1. The output from routines is in X6(and sometimes X7).

The runtime library consists of routines which execute certain elements of the FASBOL - 6000 source language(primary routines) and of routines which perform functions invisible to the user (secondary routines). The free storage system employs the use count/release stack mechanism of FASBOL although storage allocation is by the "first fit" method rather than the modified buddy system.

The runtime system uses 6 separate stacks. These are the SS,ES,AS, CV,PS, and RS stacks, representing respectively the system, expression, assignment, conditional value, pattern and release stacks. All these stacks have the same structure, which is diagrammed in the section on system tables.They are of infinite (core limit) size. SS is used to keep track of the function level and the base value of the other stacks at lower function levels. ES is used in expression evaluation, calls to user defined function calls and during pattern matching. RS is used by the free storage mechanism. AS,CV and PS are used only during pattern matching.

User defined functions and user defined datatypes operate as in FASBOL. When a function is called a function block is acquired from the pool. Values and system parameters are stored in the block in order to provide for possible re-entry of the function. Only the pointer to the block is saved on the SS stack.

TABLE 2

Descriptor Formats

STRING

4	2	18	18	18
1000	0	# of characters	Pointer to last word	Pointer to first word
DS$TYP		DS$STN	DS$SPL	DS$SPF

(String is packed 10 chars/word)

Immediate String

4	3	5	48
0000	0	#	≤ 8 characters left justified/zero fill
DS$TYP		DS$ISN	DS$ISC

Real

4	38	18
0001	0	Pointer to value
DS$TYP		DS$RPT

Integer

4	56
0010	56 bit signed binary integer
DS$TYP	DS$IVL

Name

4	4	4	6	4	2	18	18
0011	0				0	Pointer to I/O Association Pointer	Pointer to Location
DS$TYP	DS$NOB	DS$NDE		DS$NIO		DS$ASP	DS$NPL

DS$NOB
- 0 - Variable
- 1 - Label
- 2 - Function
- 3 - File

DS$NDE
- 0 - NOT Dedicated
- 1 - Dedicated String
- 2 - Dedicated Integer
- 3 - Dedicated Real
- 4 - Keyword
- 5 - Dedicated Pattern

DS$NIO
- 0 - NOT Attached (DS$ASP - 0)
- 1 - Input Attached
- 2 - Output Attached

I/O Association Pointer

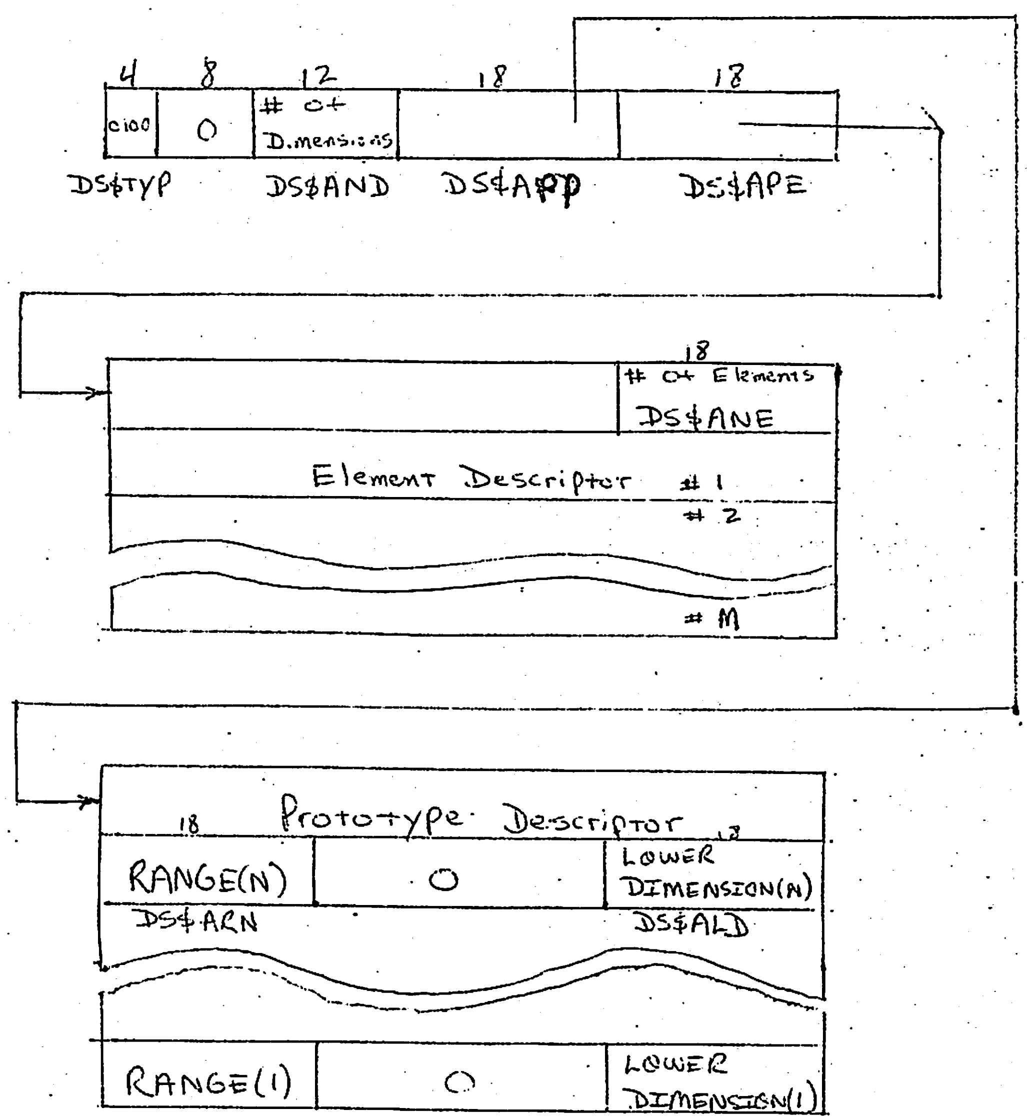
7	15	15	5	18
Carriage control character	# of chars in Rec length	# of words in Record length		Pointer to FET
AS$CCC	AS$NCH	AS$NWD	AS$NCL (# of chars in last word of Rec len)	AS$FET

Normal String Header

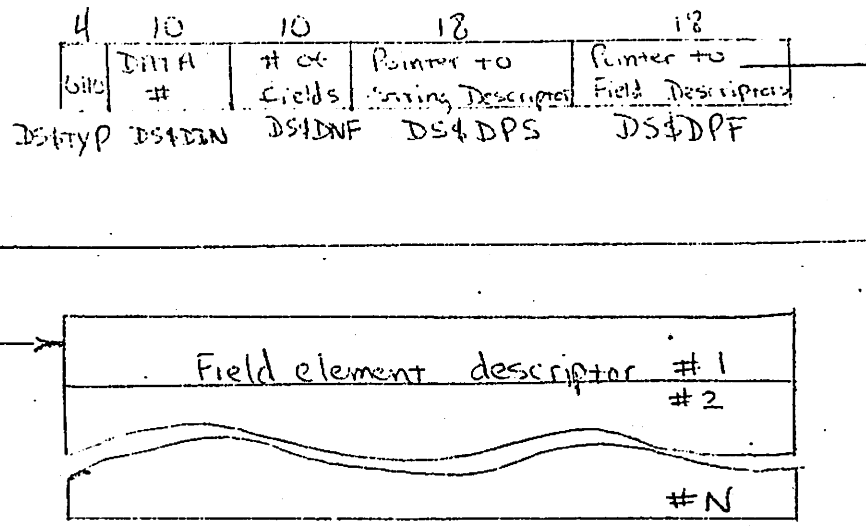
(word before start of string)

1	2	3	18	18	18
		0	# of chars in last word	# of words	USE COUNT
SH$RS (RS Flag)	SH$IUS (use flag)		SH$CHL	SH$NWR	SH$UCT

Array


4
8
12
18
18
c100
0
# of Dimensions
DS$TYP
DS$AND
DS$APP
DS$APE
18
# of Elements
DS$ANE
Element Descriptor
1
2
M
Prototype Descriptor
18
18
RANGE(N)
0
LOWER DIMENSION(N)
DS$ARN
DS$ALD
RANGE(1)
0
LOWER DIMENSION(1)

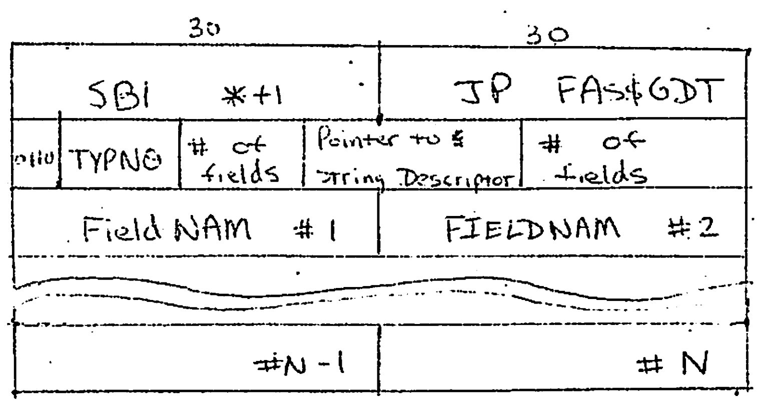
Programmer-Defined Datatype



TYPNAM :

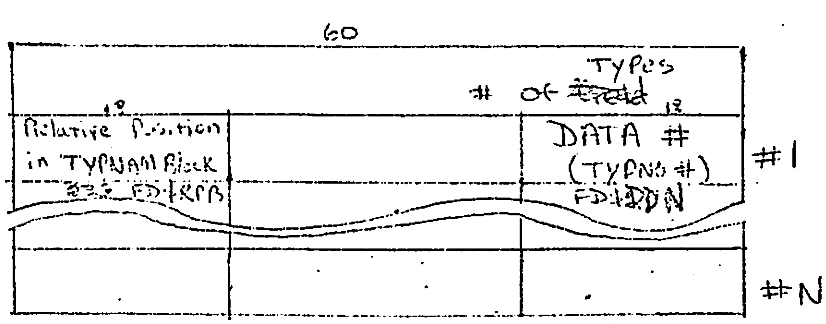
(FDEFS as above)

(Used to create an instance of the datatype)

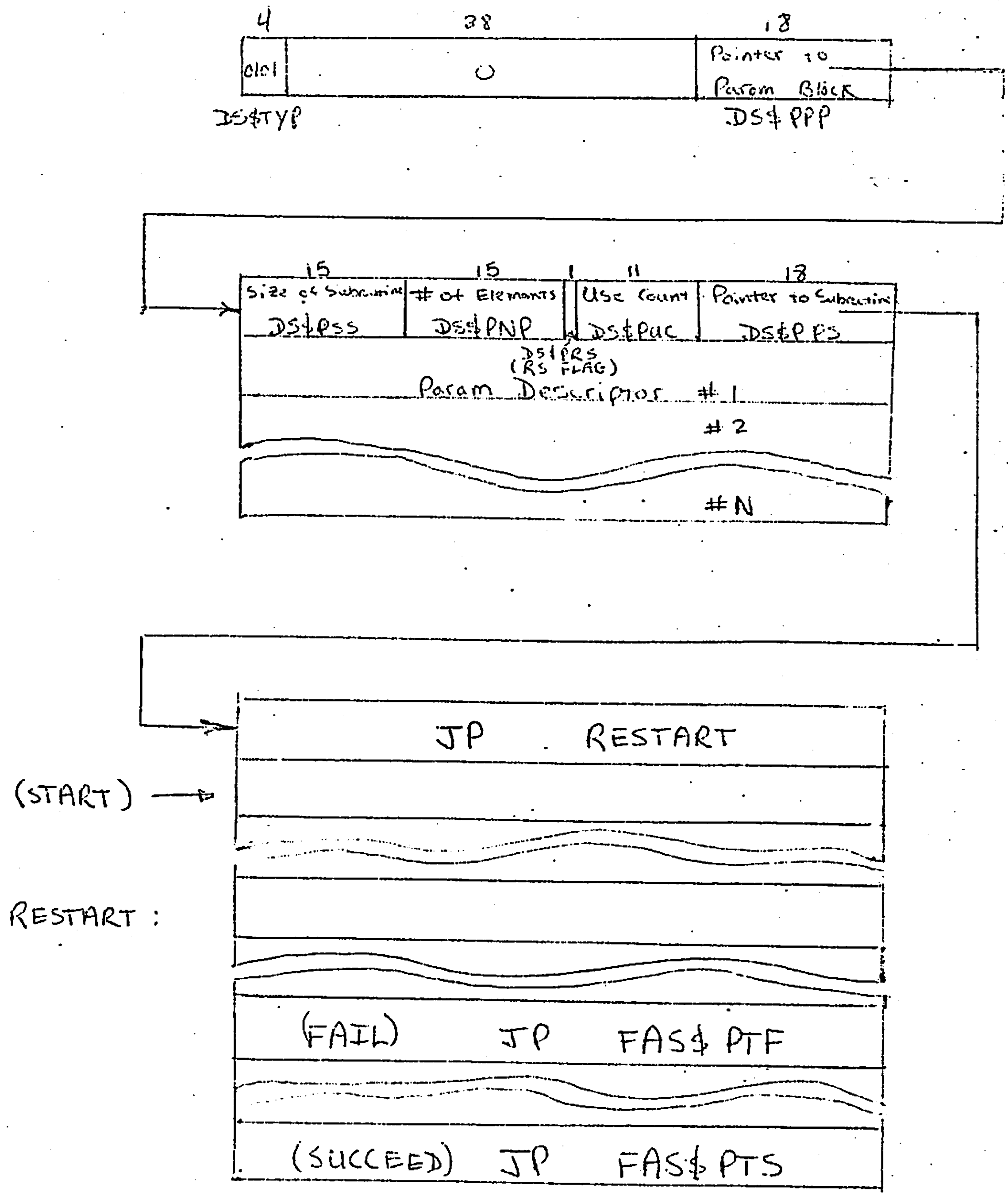


FIELDNAM:

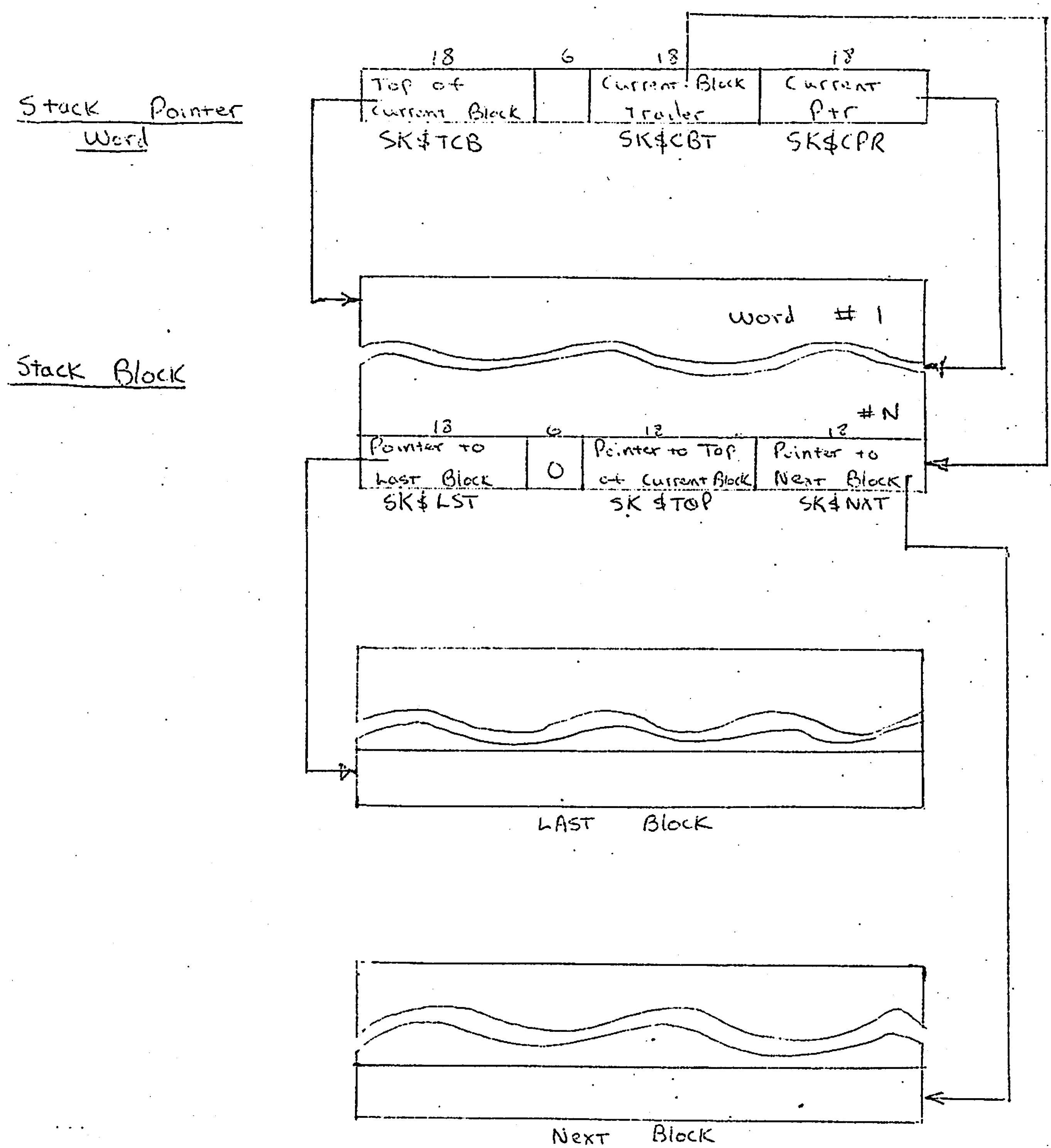
(used for field references)



Pattern


4
38
18
0101
0
Pointer to Param Block
DS$TYP
DS$PPP
15
15
1
11
18
Size of Subroutine
# of Elements
Use Count
Pointer to Subroutine
DS$PSS
DS$PNP
DS$PUC
DS$PPS
DS$PRS
(RS FLAG)
Param Descriptor #1
#2
#N
JP RESTART
(START)
RESTART :
(FAIL) JP FAS$PTF
(SUCCEED) JP FAS$PTS

Stack Formats



4. Further Developments and Improvements

4.1 Compiler

As mentioned in the introduction the compiler is currently being re-written to make it faster. These changes will also decrease the field length required for the compiler to compile itself. The compiler currently requires 117K (octal) to compile. Since the maximum field length allowed at the University Computer Center during normal operations is 120K the first priority is to make the field length requirements smaller.

The first contemplated improvement is the recognition of declarations for (a) specifying that the current program being compiled is a main program or subprogram or (b) naming external FORTRAN and FASBOL - 6000 subroutines. This would allow the compiler to be broken up into three or four subroutines and a main program which calls each successively. Then using CLDR an overlay structure could be established which would allow the compiler to run in a smaller field length. (An educated guess - 45K minimum and perhaps 70K for self-compilation.) This (and other changes to the compiler will be relatively easy to implement since the compiler is written in FASBOL - 6000.

The second planned development is a mechanism to allow the user to purge symbols from the runtime symbol table which are not referenced indirectly (with the unary $ operator). This change along with an anticipated reorganization of the way the compiler stores information about variables and literals should save about 7K (octal).

The next anticipated improvement is the compile-time recognition of pattern structures, i.e., the pattern structure

TAB(*P) ANY(↑+-/*↑) NSPAN(↑ ↑) @P

could be compiled as one pattern with four parameters. Presently it is compiled as the concatenation of four patterns. The techniques for this change are currently known but their implementation will require extra code in the compiler and hence extra field length, which is currently unavailable. This change alone should give a significant improvement in execution speed as well as savings in space since runtime concatenation of patterns is fairly slow and requires additional layers of pattern structure which also causes the resultant pattern to be larger and slower.

The fourth improvement in line is the implementation of dedicated variables. Variables in SNOBOL4 can be one of several TYPEs at runtime and their TYPE can change during the execution. This fact requires the runtime system to do a TYPE check every time it deals with a variable. This is of course a convenient and powerful feature, but one that is seldom used for most variables. If the programmer specifies to the compiler which variables' TYPEs are constant throughout the execution, much more efficient code can be generated in some cases. Specifically FORTRAN-like code can be generated for dedicated arithmetic expressions yielding up to one hundred times increase in execution speed. Once again these techniques are known and await only more usable field length.

Further developments include having the compiler produce relocatable binary rather than COMPASS object code. This would save the assembly phase which actually takes longer than the compilation phase. Eventually the compiler could be re-written in COMPASS or SYMPL to speed up the compilation time and reduce the field length requirements.

4.2 The runtime system.

The runtime system is fairly solid at this point. The only planned change other than future bug-fixing has to do with the register conventions. A0 is at present not used at all by the runtime library. In addition registers A3, X3, and B3 could be put to better use. At present these registers are used to hold information about the current pattern during pattern matching and a flag to indicate to the pattern matching mechanism that all further restarts will fail so that the match can be aborted in QUICKSCAN mode. If these are placed in core instead of registers, a memory reference will be required every time their value is needed, but the registers A3, X3, and B3 would be freed up for other use, namely for stack operations. I. E. A3 and X3 could be dedicated to use by the ES stack ~~and A0 and B3 could be dedicated to use by the ES stack~~ and A0 and B3 could be dedicated to either the PS or RS or SS stack. This would afford a savings of the equivalent of seven memory references for each PUSH of POP of these stacks. This should be an improvement since these stacks are heavily used at runtime.

The only other known improvement is to make the I/O operations asynchronous. This would save the CP time spent waiting for the PP to recognize a CIO call. It would, however, add a great deal of complexity to the I/O mechanism.

REFERENCES

1. Farber, D. J., Griswold, R. E., and Polonsky, I. P., "SNOBOL, A String Manipulation Language," Journal of the ACM, 11, 1 (1964).

2. ..., ..., and ..., "The SNOBOL3 Programming Language," Bell System Technical Journal, 45, 6 (1966).

3. Forte, A., SNOBOL3 Primer, The M.I.T. Press, (1967).

4. Griswold, R. E., Poage, J. F., and Polonsky, I. P., The SNOBOL4 Programming Language, Prentice-Hall, (1968), (First Edition).

5. ..., ..., and ..., The SNOBOL4 Programming Language, Prentice-Hall (1971), (Second Edition).

6. Shapiro, M. D., "A Bibliography of SNOBOL Publications," SIGPLAN Notices of the ACM, 4, 11, (1969), pp. 41-48.

7. Dewar, B. K., and Belcher, K., "SPITBOL," SIGPLAN Notices of the ACM, 4, 11, (1969), pp. 33-35.

8. Santos, P. J., FASBOL, A SNOBOL Compiler, Ph.D. Thesis, University of California, Berkeley, also Memorandum No. ERL - M314, Electronics Research Laboratory, University of California, Berkeley, December 1971.

9. Santos, P. J., FASBOL II, A SNOBOL4 Compiler for the PDP-10, Memorandum No. ERL - M348, Electronics Research Laboratory, University of California, Berkeley, August, 1972.

10. Gaines, R. S., Preliminary Report on the SNOBOL4 Programming Language Revised to Conform to the CDC 6600 Implementation, Institute for Defense Analyses, March, 1968.

11. CAL SNOBOL, Code LO, Computer Center, University of California, Berkeley, April, 1971. (no author given).

APPENDIX I

Listing of the FASBOL-6000 Compiler

```
*
*                        FIRST EXECUTABLE STATEMENT
*
               STTIM      = TIME()
               E          = ↑=↑
               S          = ↑;↑
               Q          = ↑:↑
               ^ANCHOR    = 1
                 DEFINE(↑OPENOB(D)↑)
                 DEFINE(↑INISYM(D)↑)
                 DEFINE(↑SPCLIN(PAR)↑)
                 DEFINE(↑PRTLIN(PAR)↑)
                 DEFINE(↑EJECT(D)↑)
                 DEFINE(↑GETSA(D)↑)
                 DEFINE(↑PUTOUT(PAR)↑)
                 DEFINE(↑LABEL(LAB)↑)
               DEFINE(↑VAR(PAR,REG)↑)
               DEFINE(↑VARNM(PAR)↑)
               DEFINE(↑FUNC(FUN,REG)↑)
               DEFINE(↑FUNCNAME(FUN,REG)↑)
                 DEFINE(↑ERRMSG(PAR)↑)
                 DEFINE(↑FNC(P1,P2,P3,P4)N↑)
                  DEFINE(↑WORDS(P)I,J↑)
               DEFINE(↑CONSTANT(CON)↑)
               DEFINE(↑TOP()↑)
               DEFINE(↑PUSH(D,INP)↑)
               DEFINE(↑POP(D)↑)
               DEFINE(≠PARAM(END)≠)
               DEFINE(↑CALL(ROU)↑)
               DEFINE(↑CALL1(ROU)↑)
               DEFINE(↑EXPRESSION()STACK,STATE,NEXTSTATE,VARI,ALTNUM,IND↑)
               DEFINE(↑DUMP(STR)↑)
               DEFINE(↑GOTOP(LAB)↑)
               DEFINE(↑NEWLAB(C)↑)
               DEFINE(↑NEXTLAB(D)↑)
               DEFINE(↑G.EX(D)↑)
               DEFINE(↑GETLIN(D)↑)
               DEFINE(↑USE(STR)↑)
               DEFINE(↑FNCBLNM(D)↑)
               DEFINE(↑PDF(PAR)↑)
               DEFINE(↑FIELD(FD)↑)
               DEFINE(↑VALUE(REG)VARN↑)
               DEFINE(↑NAME(REG)↑)
               DEFINE(↑PATVAR()↑)
               DATA(↑FDATA(TYPEND,LAST)↑)
               DATA(↑VARID(CON,TYP)↑)
               PCPRMT     = ↑(↑ NSPAN(↑ ↑)
               PCPRMT     = NSPAN(↑ ↑) ↑)↑
               LNO        = 10000
               NUMBER     = (ANY(↑+-↑) v NULL) SPAN(↑0123456789↑)
               SPB        = SPAN(↑ ↑)
               NSP        = NSPAN(↑ ↑)
               LISTSR     = 1
```

```
                PROG         = ↑MAIN↑
                DIAGNO       = 0
                IDENPT       = ANY(↑ABCDEFGHIJKLMNOPQRSTUVWXYZ↑)
                NSPAN(↑ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-$.↑)
                SP7          = ↑ ↑
                SP8          = ↑  ↑
                SP10         = ↑    ↑
                BX1X6        = SP10 ↑BX1        X6↑
                BX6X1        = SP10 ↑BX6        X1↑
                USEGOT       = ↑   USE GOTO↑
                USEAST       = ↑   USE *↑
                ASMFLG       = 1
                &STLIMIT = 1000000
                CN           = ARRAY(↑10001/11001,1/3↑)
                FX           = ARRAY(500)
                INDENT       = ↑        ↑
                SEMIPT       = REALCH(↑;↑)
                INTGPT       = NSPAN(↑0↑) SPAN(↑0123456789↑) $ INTGER
                REAL         = NUMBER ↑.↑ NSPAN(↑0123456789↑)
                NXTLIN       = GETLIN()
                LIST.        = ↑C.1↑
                UNLIST.      = ↑C.2↑
                NOCODE.      = ↑C.3↑
                CODE.        = ↑C.4↑
                EJECT.       = ↑C.5↑
                FAIL.        = ↑C.6↑
                NOFAIL.      = ↑C.7↑
                NOCROSS.     = ↑C.8↑
                CROSSREF.     = ↑C.9↑
                NOASM.       = ↑C.10↑
                LISTASM.     = ↑C.11↑
                SPACE.       = ↑C.12↑
                NEWSTNO.     = ↑C.13↑
                SNAP.        = ↑C.14↑
                CSNAP.       = ≠C.15≠
                EX           = E  ↑XFAS$↑
                JP1          = ↑      JP↑ SP8 EX
                JP = ↑       ↑ JP1
                SA61         = ↑SA6↑ SP7 EX
                SA6          = SP10 SA61
                RJ           = ↑      RJ↑ SP8 EX
                PTF          = ≠   NZ B1,=XFAS$PTF≠
                PTS          = ↑   JP =XFAS$PTS↑
      *
      *                      INITIALIZATION OF FSPDM FOR EXPRESSION RECOGNITION
      *
                NSF          = ARRAY(↑3,6↑)
                NSF[1,1]     = 2
                NSF[1,2]     = 2
                NSF[1,3]     = 2
                NSF[2,2]     = 3
                NSF[3,2]     = 4
                NSF[1,4]     = 5
                NSF[1,5]     = 5
                NSF[1,6]     = 5
                NSF[2,5]     = 6
```

```
103      NSF[3,5]   = 4
104      NAF        = ARRAY(↑6,3,5↑)
105      NAF[1,1,1]  = ↑A1↑
106      NAF[2,1,1]  = ↑A3↑
107      NAF[2,2,1]  = ↑A2↑
108      NAF[2,1,2]  = ↑A4↑
109      NAF[2,2,2]  = ↑A2↑
110      NAF[2,1,3]  = ↑A5↑
111      NAF[2,2,3]  = ↑A7↑
112      NAF[3,1,3]  = ↑A1↑
113      NAF[2,1,4]  = ↑A5↑
114      NAF[2,2,4]  = ↑A6↑
115      NAF[3,1,4]  = ↑A1↑
116      NAF[2,3,1] = ↑A8↑
117      NAF[2,3,2] = ↑A9↑
118      NAF[2,3,3] = ↑A10↑
119      NAF[2,3,4] = ↑A10↑
120      NAF[4,1,5] = ↑A1↑
121      NAF[5,1,5] = ↑A3↑
122      NAF[5,1,2] = ↑A4↑
123      NAF[5,1,3] = ↑A5↑
124      NAF[6,1,3] = ↑A1↑
125      NAF[5,1,4] = ↑A5↑
126      NAF[6,1,4] = ↑A1↑
127      NAF[5,2,5] = ↑A2↑
128      NAF[5,2,2] = ↑A2↑
129      NAF[5,2,3] = ↑A7↑
130      NAF[5,2,4] = ↑A6↑
131      NAF[5,3,5] = ↑A11↑
132      NAF[5,3,2] = ↑A12↑
133      NAF[5,3,3] = ↑A13↑
134      NAF[5,3,4] = ↑A13↑
135      $(↑+↑)      = ↑AAD↑
136      $(↑-↑)      = ↑ASB↑
137      $(↑*↑)      = ↑AML↑
138      $(↑/↑)      = ↑ADV↑
     *
     *                    PROCESSING SEQUENCE
     *
     *                    INITIALIZE SYMBOL TABLES
     *
139      INISYM()
     *
     *                    OPEN SOURCE AND OBJECT FILES
     *
140      OPENOB()                                        :(PROCPH)
     *
     *
     *
     *                    EXECUTABLE STATEMENT PHASE
     *
     *
     *                    END ACTION PHASE
     *
     *
     *                    END PROCESSING
```

```
      *
  141 ENDMSG   IF(SPCLIN(6) PRTLIN(↑ *TOTAL COMPILATION TIME↑ Q
  141 .    TIME() - STTIM ↑ MS., ↑ DIAGNO ↑ ERROR DIAGNOSTICS*↑) EJECT())
  141 .                                                         :(END)
      *                       OPENOB FUNCTION
      *
  142 OPENOB      OUTPUT(↑PUT↑,↑ASM↑)
  143             OUTPUT(↑PAGE↑,↑OUTPUT↑,↑1↑)
  144             PAGE      = ↑FASBOL COMPILER VER 0.2 ↑ DATE() ↑     ↑ CLOCK()
  145             SPCLIN(2)
  146             PUTOUT(SP10 ↑IDENT       ↑ PROG)
  147             PUTOUT(SP10 ↑ENTRY       ↑ PROG)
  148             PUTOUT(SP10 ↑EXT         FAS$FFN,FAS$SFN,FAS$END↑)
  149             PUTOUT(SP10 ↑EXT         FAS$OAP,FAS$IAF,FAS$ERR↑)
  150             PUTOUT(PROG ↑        BSS        0↑)
  151             PUTOUT(SP10 ↑SB1         PLIST↑)
  152             CALL(↑ILZ↑)                                   :(RETURN)
      *
      *                       PROCPH FUNCTION
      *
  153 PROCPH      C         = TRIM(GETSA())                     :F(NOEND)
  154             DIFFER(C)                                     :S(LABL)
      *
      *                       MAIN STATEMENT LOOP
      *
  155 STLOOP      C         = TRIM(GETSA())                     :F(NOEND)
  156             DIFFER(C)                                    :F(STLOOP)
      *
      *                       PROCESS LABEL
      *
  157 LABL        LAB       =
  158             C         (NOTANY(↑ ↑) (BREAK(↑ ↑) v RTAB(0))) . LAB =
  159             L         = LABEL(LAB)                        :F(LABL1)
  160             DIFFER(LAB,↑END↑)                             :F(ENDA)
  161             PUTOUT(L ↑     BSS        0↑)
  162 LABL1       DIFFER(C)                                     :F(STLOOP)
  163 GOTO        FAILGOT   =
  164             SUCGOT    =
  165             C         REALCH(↑:↑) . BODYS ↑:↑ =           :F(BODY)
  166             C         POPRMT =                            :F(GOTO2)
      *                     UNCONDITIONAL GOTO
  167             SUCGOT    = G.EX()                            :F(BADGOT)
  168             FAILGOT   = SUCGOT                            :(GOTO7)
  169 GOTO2       C ↑S(↑ NSPAN(↑ ↑) =                           :F(GOTO3)
      *                     SUCCESS GOTO
  170             SUCGOT    = G.EX()                            :F(BADGOT)
  171             C         POPRMT NSPAN(↑ ↑) =                 :F(BADGOT)
  172             DIFFER(C)                                     :F(GOTO8)
  173 GOTO3       C ↑F(↑ NSPAN(↑ ↑) =                           :F(BADGOT)
      *                     FAILURE GOTO
  174             FAILGOT   = G.EX()                            :F(BADGOT)
  175             IDENT(SUCGOT)                                 :F(GOTO7)
  176             C         POPRMT NSPAN(↑ ↑) =                 :F(BADGOT)
  177             DIFFER(C)                                     :F(GOTO8)
  178             DIFFER(SUCGOT)                                :F(GOTO2)S(BADGOT)
      *                     ASSURE AT END AFTER ↑)↑
```

```
GOTO7     C           PCPRMT RPOS(0)                        :F(BADGOT)
GOTO8     C           = TRIM(BODYS)
          DIFFER(C)                                         :F(STATEND)
BODY      FAILGOT     =  IDENT(FAILGOT) NEXTLAB()
          PUTOUT(SP10 ↑SB1          ↑ FAILGOT)
          PUTOUT(SP10 ↑SB2          ↑ STNO)
          CALL(↑STE↑)
          INDIR       =
          C           SPAN(↑ ↑) =                           :F(SYNTAX)
          C           REALCH(↑=↑)                           :F(MATCH)
          NAME(6)
ACP1      C           SPAN(↑ ↑) =                           :F(SYNTAX)
          C           ↑=↑ =                                 :F(MATREPL)
          PUTOUT(SA6 ↑ACP↑)
          EXPRESSION()
          IDENT(C)                                          :F(SYNTAX)
ASSIGN    CALL(↑ASG↑)
STATEND   DIFFER(SUCGOT) PUTOUT(SP10 ↑JP          ↑ SUCGOT) :(STLOOP)
MATCH     VALUE(1)
SUB1      DIFFER(C)                                         :F(STATEND)
          CALL(↑LES↑)
          EXPRESSION()
          PUTOUT(BX6X1)
          PATNUM      = PATNUM + 1
          PUTOUT(SP10 ↑SA6          D↑ PATNUM ↑+1↑)
          PUTOUT(SP10 ↑SA3          D↑ PATNUM)
          CALL(↑MTP↑)
          CALL(≠MCM≠)
          USE(↑DATA↑)
          PUTOUT(↑D↑ PATNUM ↑     VFD       30/1,30/R↑ PATNUM ↑-1↑)
          PUTOUT(≠   BSSZ 1≠)
          USE(↑*↑)
          USE(↑SUB↑)
          PUTOUT(↑R↑ PATNUM ↑     SA1       D↑ PATNUM ↑+1↑)
          CALL(↑PTP↑)
          PUTOUT(≠   NZ B1,=XFAS$MTF≠)
          PUTOUT(≠   JP =XFAS$MTS≠)
          USE(↑*↑)                                          :(STATEND)
MATREPL   PUTOUT(BX1X6)
          CALL(↑SRS↑)
          EXPRESSION()
          C           NSP ↑=↑ NSP =                         :F(SYNTAX)
          PUTOUT(BX6X1)
          PATNUM      = PATNUM + 1
          PUTOUT(SP10 ↑SA6          D↑ PATNUM ↑+1↑)
          PUTOUT(SP10 ↑SA3          D↑ PATNUM)
          CALL(↑MTP↑)
          CALL(↑SSJ↑)
          EXPRESSION()
          CALL(↑MTR↑)
          USE(↑DATA↑)
          PUTOUT(↑D↑ PATNUM ↑     VFD       30/1,30/R↑ PATNUM ↑-1↑)
          PUTOUT(≠   BSSZ 1≠)
          USE(↑*↑)
          USE(↑SUB↑)
          PUTOUT(↑R↑ PATNUM ↑     SA1       D↑ PATNUM ↑+1↑)
```

```
235                 CALL(↑PTP↑)
236                 PUTOUT(≠  NZ B1,=XFAS$MTF≠)
237                 PUTOUT(≠  JP =XFAS$MTS≠)
238                 USE(↑*↑)                                          :(STATEND)
239 DUMP            OUTPUT      = STR ↑ = ↑ $STR                      :(RETURN)
    *
    *                           G.EX FUNCTION - PARSES GOTO↑S
    *
240 G.EX            C           IDENPT  . GOT =                       :F(G.EX2)
241 G.EX1           G.EX        = GOTOP(GOT)                          :(RETURN)
242 G.EX2           C           ↑$↑ =                                 :F(FRETURN)
243                 C           ((↑≠↑ BREAK(↑≠↑) . GOT ↑≠↑) V (≠↑≠ BREAK(≠↑≠) . GOT
243 .      ≠↑≠)) =                                                    :S(G.EX1)
244                 PUTOUT(USEGOT)
245                 G.EX        = NEWLAB()
246                 PUTOUT(G.EX ↑     BSS        0↑)
247                 EXPRESSION()                                      :F(FRETURN)
248                 PUTOUT(SP10 ↑SA2         =XFAS$LBN↑)
249                 CALL(↑SYM↑)
250                 PUTOUT(SP10 ↑SB1         X6↑)
251                 PUTOUT(SP10 ↑JP          B1↑)
252                 PUTOUT(USEAST)                                    :(RETURN)
    *                           NEWLAB FUNCTION
253 NEWLAB          LNO         = LNO + 1
254                 NEWLAB      = ↑L↑ LNO                             :(RETURN)
    *                           NEXTLAB FUNCTION
255 NEXTLAB         NEXTLABL    = NEWLAB()
256                 NEXTLAB     = NEXTLABL                            :(RETURN)
257 GOTOP           GOTOP       = DIFFER($(LAB ↑$L1↑)) $(LAB ↑$L1↑)   :S(RETURN)
258                 GOTOP       = DIFFER($(LAB ↑$L2↑)) $(LAB ↑$L2↑)   :S(RETURN)
259                 GOTOP       = NEWLAB()
260                 $(LAB ↑$L2↑) = GOTOP
261                 LABNUM      = LABNUM + 1
262                 $(↑L↑ LABNUM) = LAB                               :(RETURN)
    *
    *
    *                           VALUE FUNCTION
    *
263 VALUE           C           ↑$↑ =                                 :S(INDV)
264                 C           ↑+↑ =                                 :S(VALUE)
265                 C           ↑-↑ =                                 :S(NEGV)
266                 C           ↑(↑ =                                 :S(EXPV)
267                 C           ↑.↑ =                                 :S(NAMV)
268                 C           ↑^↑ =                                 :S(KEYNV)
269                 C           IDENPT $ VARN =                       :S(VARV)
270                 C           ANY(↑≠↑ ≠↑≠) $ DEL =                  :S(LITV)
271                 C           NUMBER $ CON =                        :S(INTV)
271 .   F(NULL)
272 INDV            VALUE(1)
273                 CALL(↑IVV↑)
274 FIN6            RR          = 6
275 FIN             NE(RR,REG)                                        :F(RETURN)
276                 PUTOUT(SP10 ↑BX↑ REG SP7 ↑X↑ RF)                  :(RETURN)
277 NEGV            VALUE(1)
278                 CALL(↑ANG↑)                                       :(FIN6)
279 EXPV            EXPRESSION()
```

```
280                C             NSPAN(↑ ↑) ↑)↑ =                      :F(UNBAL)
281 FINI           RR            = 1                                   :(FIN)
282 NAMV           NAME(REG)                                           :(RETURN)
283 KEYWV          C             IDENPT $ KEY =                        :F(KEYERR)
284                KEY           = $(KEY ↑.↑)
285                DIFFER(KEY)                                         :F(KEYERR)
286                PUTOUT(↑  SB1  ↑ KEY ↑B↑)
287                CALL(↑FKW↑)                                         :(FIN6)
288 VARV           C             ↑(↑ =                                 :S(FUNV)
289                C             ↑[↑ =                                 :S(ARV)
290                VAR(VARN,1)
291                CALL(↑VAL↑)                                         :(FIN6)
292 LITV           C             BREAK(DEL) $ CON DEL =                :F(NOFOOT)
293 CONV           CINDEX        = 1
294                LE(REG,5)                                           :F(OUTRL)
295                PUTOUT(SP10 ↑SA↑ REG SP7 CONSTANT(CON))               :(RETURN)
296 OUTRL          PUTOUT(SP10 ↑SA1        ↑ CONSTANT(CON))               :(FIN1)
297 INTV           C             ↑.↑ SPAN(↑0123456789↑) $ RPT =        :F(CONV)
298                CON           = CON ↑.↑ RPT                         :(CONV)
299 NULLV          VALUE(1)
300                PUTOUT(SP10 ↑BX↑ REG SP7 ↑XU-XO↑)                   :(RETURN)
301 FUNV           FUNC(VARN,REG)                                      :(RETURN)
302 ARV            PUTOUT(SP10 ↑SB1         ↑ PARAM(↑]↑))
303                VAR(VARN,2)
304                PUTOUT(SP10 ↑SA2         X2↑)
305                CALL(↑RAR↑)
306 NAM6           LE(REG,5)                                           :F(OUTRA)
307                PUTOUT(SP10 ↑SA↑ REG SP7 ↑X6↑)                      :(RETURN)
308 OUTRA          PUTOUT(SP10 ↑SA1         X6↑)                       :(FIN1)
    *
    *                            NAME FUNCTION
    *
309 NAME           C             ↑$↑ =                                 :S(INDN)
310                C             ↑^↑ =                                 :S(KEYN)
311                C             IDENPT $ NAM =                        :F(NONAME)
312                C             ↑(↑ =                                 :S(FUNN)
313                C             ↑[↑ =                                 :S(ARN)
314                VAR(NAM,REG)                                        :(RETURN)
315 FUNN           FUNCNAME(NAM,REG)                                   :(RETURN)
316 INDN           VALUE(1)
317                CALL(↑IVN↑)                                         :(FIN6)
318 KEYN           C             IDENPT $ K =                          :F(KEYERR)
319                KEY           = $(K ↑.↑)
320                DIFFER(KEY)                                         :F(KEYERR)
321                K             = E ↑14000400000000000↑ LPAD(KEY,2,↑0↑) ↑B↑
322                LE(REG,5)                                           :F(KNOUTP)
323                PUTOUT(↑  SA↑ REG ↑ ↑ K)                            :(RETURN)
324 KNOUTP         PUTOUT(↑  SA1 ↑ K)                                 :(FIN1)
325 ARN            PUTOUT(SP10 ↑SB1         ↑ PARAM(↑]↑))
326                VAR(NAM,2)
327                PUTOUT(SP10 ↑SA2         X2↑)
328                CALL(↑RAR↑)                                         :(FIN6)
    *                            EXPRESSION FUNCTION
    *
329 EXPRESSION  STATE        = 1
330                STACK         =
```

```
331 EXPLOOP   IND          =
332           OPER         = 1
333           C            SPB =
334           DIFFER(C)                                       :F(EXPEND)
335 DOLLAR    C            ↑$↑ =                              :S(INDR)
336           C            ↑(↑ =                              :S(PAREN)
337           C            ↑≠↑ =                              :S(CON1)
338           C            ≠↑≠ =                              :S(CON2)
339           C            NUMBER . CON =                     :S(CON6)
340           C            ↑^↑ =                              :S(KEYW)
341           C            ↑.↑ =                              :S(EXPNAME)
342           C            IDENPT . VARI =                    :F(OP)
343           C            ↑(↑ =                              :S(EXPFUNC)
344           L            = ↑VARIB↑                          :(NEXTACT)
345 VARIB     C            ↑[↑ =                              :S(EXPARRY)
346           VAR(VARI,1)
347           IDENT(PVI)                                      :F(VRB1)
348           CALL1(↑VAL↑)
349 VRB1      PVI          =
350 TESTIND   GT(IND)                                         :F(PATASTST)
351           CALL1(↑IVV↑)
352           IND          = IND - 1                          :(TESTIND)
353 OP        C            ANY(↑+-*/↑) $ OPTOR ↑ ↑ =          :F(ALT)
354           OPER         = 2                                :(NEXTACT)
355 ALT       C            ↑v ↑ =                             :S(ALT1)
356           C            ↑// ↑ =                            :F(EXPEND)
357 ALT1      OPER         = 3
358 NEXTACT   NEXTSTATE    = NSF[OPER,STATE]
359           ACT          = DIFFER(NEXTSTATE) NAF[STATE,OPER,TOP(STACK)]
359 .                                                         :F(SYNTAX)
360           STATE        = DIFFER(ACT) NEXTSTATE            :S($ACT)F(SYSERR)
361 INDR      IND          = IND + 1
362           C            SPB =                              :(DOLLAR)
363 PAREN     L            = ↑PAREN1↑                         :(NEXTACT)
364 PAREN1    EXPRESSION()
365           C            SPB =
366           C            ↑)↑ =                              :F(MISPAR)S(TESTIND)
367 CON1      C            BREAK(↑≠↑) . CON ↑≠↑ =             :F(MISQUOTE)S(CON3)
368 CON2      C            BREAK(≠↑≠) . CON ≠↑≠ =             :F(MISQUOTE)
369 CON3      CINDEX       = 1
370 CON3A     L            = ↑CON4↑                           :(NEXTACT)
371 CON4      PUTOUT(SP10 ↑SA1       ↑ CONSTANT(CON))          :(TESTIND)
372 CON6      CINDEX       = 2
373           C            ↑.↑ NSPAN(↑0123456789↑) $ CO =      :F(CON3A)
374           CINDEX       = 3
375           CON          = CON ↑.↑ CO                       :(CON3A)
376 EXPARRY   PUTOUT(SP10 ↑SB1       ↑ PARAM(↑]↑))
377           VAR(VARI,2)
378           PUTOUT(SP10 ↑SA2       X2↑)
379           CALL(↑RAR↑)
380           PUTOUT(SP10 ↑SA1       X6↑)                      :(TESTIND)
381 EXPFUNC   DIFFER($(VARI ↑$F3↑))                           :S(NULRET)
382           L            = ↑EXPFUNC2↑                       :(NEXTACT)
383 NULRET    FUNC(VARI,1)
384           IDENT(IND)                                      :S(EXPLOOP)F(SYNTAX)
385 EXPFUNC2  FUNC(VARI,1)                                    :(TESTIND)
```

```
386 EXPNAME     L            = *EXNAM2*                               :(NEXTACT)
387 EXNAM2      NAME(1)                                               :(TESTING)
388 EXPEND      EQ(STATE,1)                                           :S(NULL)
389           IF(NE(STATE,3) NE(STATE,4) NE(STATE,6))                 :F(SYNTAX)
390           . GT(STATE,3)                                           :F(EXENLOOP)S($(*AL*
390 .     TOP())
391 AL3
392 AL4         CC           = POP()
393 AL4A        CALL($CC)
394 AL2         CC           = POP()                                  :F(AL5)
395             PUTOUT(BX1X6)
396             CC           *C* =                                    :F(AL4A)
397             CALL(*LES*)
398             PUTOUT(SP10 *SB1         * CC)
399             CALL(*CNC*)                                           :(AL5A)
400 AL5         PUTOUT(BX6X1)
401 AL5A        V            = POP()                                  :F(SYSERR)
402             V            LEN(1) REM $ V
403             V            = V + 1
404             PUTOUT(SP10 *SA6         D* ALTNUM *+* V)
405             USE(*ALT*)
406             PUTOUT(SP10 *SA1         D* ALTNUM *+* V)
407             CALL(*PTP*)
408             PUTOUT(SP10 *NZ          B1,FL* ALTNUM)
409             ALTP         = ALTP + 1
410             PUTOUT(SP10 *SX1         A* ALTP)
411             PUTOUT(SP10 *JP          CP* ALTNUM)
412             PUTOUT(*A* ALTP *     SA1         D* ALTNUM *+* V)
413             PUTOUT(BX6X1)
414             PUTOUT(SP10 *AX6         56*)
415             PUTOUT(SP10 *SX6         X6-5*)
416             PUTOUT(SP10 *NZ          X6,FL* ALTNUM)
417             CALL(*RPT*)
418             PUTOUT(SP10 *NZ          B1,FL* ALTNUM)
419             PUTOUT(SP10 *SX1         A* ALTP)
420             USE(*PAT*)
421             PUTOUT(*FL* ALTNUM RJ *UES*)
422             PUTOUT(PTF)
423             PUTOUT(*RS* ALTNUM RJ *UES*)
424             PUTOUT(SP10 *SB1         X6*)
425             PUTOUT(SP10 *JP          B1*)
426             PUTOUT(*P* ALTNUM *      VFD         4/5,56/D* ALTNUM)
427             USE(***)
428             USE(*DATA*)
429             PUTOUT(*D* ALTNUM *      VFD         30/* V *,30/R* ALTNUM)
430             PUTOUT(SP10 *BSSZ        * V)
431             USE(***)
432             PUTOUT(*CP* ALTNUM RJ *LES*)
433             PUTOUT(SP10 *SB3         B0*)
434             PUTOUT(PTS)
435             USE(***)
436             PUTOUT(SP10 *SA1         P* ALTNUM)
437             POP()                                                 :S(SYNTAX)F(RETURN)
438 EXENLOOP    OP           = POP(STACK)                             :F(RETURN)
439             OP           *C* =                                    :S(CAT)
440             CALL1($OP)                                            :(EXENLOOP)
```

```
CAT       CALL(↑LES↑)
          PUTOUT(SP10 ↑SB1        ↑ OP)
          CALL1(↑CNC↑)
          POP(STACK)                                 :F(RETURN)S(SYSERR)
NULL      PUTOUT(SP10 ↑BX1        X1-X1↑)            :(RETURN)
KEYE      C           IDENPT $ K =                   :F(KEYERR)
          KEY        = $(K ↑.↑)
          L          = DIFFER(KEY) ↑KEYR↑            :F(KEYERR)S(NEXTACT)
KEYR      PUTOUT(↑  SB1  ↑ KEY ↑B↑)
          CALL1(↑FKW↑)                               :(TESTIND)
PATASTST  C          SPAN(↑ ↑) ANY(↑$.↑) $ PAS SPAN(↑ ↑) =  :F(EXPLOOP)
          PATNUM     = PATNUM + 1
          PUTOUT(BX6X1)
          PUTOUT(SP10 ↑SA6        D↑ PATNUM ↑+1↑)
          PUTOUT(SP10 ↑SA1        F↑ PATNUM)
          USE(↑DATA↑)
          PUTOUT(↑D↑ PATNUM ↑     VFD        30/1,30/F↑ PATNUM)
          PUTOUT(SP10 ↑BSSZ       1↑)
          USE(↑*↑)
          USE(↑PAT↑)
          PUTOUT(↑P↑ PATNUM ↑     VFD        4/5,56/D↑ PATNUM)
          PUTOUT(↑FL↑ PATNUM RJ ↑UAS↑)
          PUTOUT(PTF)
          USE(↑*↑)
          USE(↑SUB↑)
          PUTOUT(↑R↑ PATNUM ↑     JP         RS↑ PATNUM)
          CALL(↑LCA↑)
          PUTOUT(SP10 ↑SA1        D↑ PATNUM ↑+1↑)
          CALL(↑PTP↑)
          PUTOUT(SP10 ↑NZ         B1,FL↑ PATNUM)
          USE(↑PAT↑)
          RN         = ↑RCA↑
          RN         = IDENT(PAS,↑$↑) ↑RIA↑
          PUTOUT(↑PS↑ PATNUM ↑    SA1        D↑ PATNUM ↑+1↑)
          PUTOUT(BX6X1)
          PUTOUT(SP10 ↑AX6        56↑)
          PUTOUT(SP10 ↑SX6        X6-5↑)
          PUTOUT(SP10 ↑NZ↑ SP8 ↑X6,↑ EX ↑PTF↑)
          CALL(RN)
          PUTOUT(SP10 ↑SA1        D↑ PATNUM ↑+1↑)
          CALL(↑RPT↑)
          USE(↑*↑)
          NAME(2)
          RN         = ↑CPA↑
          RN         = IDENT(PAS,↑$↑) ↑IPA↑
          CALL(RN)
          PUTOUT(JP ↑PTS↑)
          USE(↑*↑)                                   :(EXPLOOP)
A1                                                   :($L)
A2        CALL(↑LES↑)
          PUSH(STACK,OPTOR)                          :(EXPLOOP)
A3        CC         = 1                             :(A4A)
A4        CC         = POP(STACK)
          CC         ↑C↑ =
A4A       CALL(↑LES↑)
          PUSH(STACK,↑C↑ CC + 1)                     :($L)
```

```
A5      OP        = POP(STACK)
        CALL1($OP)
A5LOOP  CC        = POP(STACK)                                     :F(A3)
        CC        ↑C↑ =                                            :S(A4A)
        CALL1($CC)                                                 :(A5LOOP)
A6      OP        = POP(STACK)
        CALL1($OP)                                                 :(A2)
A7      OPTOP     ANY(↑+-↑)                                        :S(A6)F(A2)
A8      PUTOUT(BX6X1)
A8A     V         = 1
        PATNUM    = PATNUM + 1
        ALTNUM    = PATNUM
        PUTOUT(SP10 ↑SA6          D↑ ALTNUM ↑+↑ V)
        USE(↑ALT↑)
        PUTOUT(↑R↑ ALTNUM ↑    JP         RS↑ ALTNUM)
        CALL(↑LCR↑)
A8B     PUTOUT(SP10 ↑SA1          D↑ ALTNUM ↑+↑ V)
        CALL(↑PTP↑)
        PUTOUT(SP10 ↑NZ           B1,*+5↑)
        ALTP      = ALTP + 1
        PUTOUT(SP10 ↑SX1.         A↑ ALTP)
        PUTOUT(SP10 ↑JP           CP↑ ALTNUM)
        PUTOUT(↑A↑ ALTP ↑     SA1        D↑ ALTNUM ↑+↑ V)
        CALL(↑PRT↑)
        PUTOUT(SP10 ↑NZ           B1,*+2↑)
        PUTOUT(SP10 ↑SX1          A↑ ALTP)
        PUTOUT(SP10 ↑JP           CP↑ ALTNUM)
        CALL(↑RCR↑)
        PUSH(,↑V↑ V)
        USE(↑*↑)                                                   :(EXPLOOP)
A9      CC        = POP()
        CC        LEN(1) REM $ CC
A9A     CALL(↑LES↑)
        PUTOUT(SP10 ↑SB1          ↑ CC)
        CALL(↑CNC↑)                                                :(A8A)
A10     OP        = POP()
        CALL($OP)
A10LOOP CC        = POP()                                          :F(A8A)
        PUTOUT(BX1X6)
        CC        ↑C↑ =                                            :S(A9A)
        CALL($CC)                                                  :(A10LOOP)
A11     PUTOUT(BX6X1)
A11A    V         = POP()
        V         LEN(1) REM $ V
        V         = V + 1
        PUTOUT(SP10 ↑SA6          D↑ ALTNUM ↑+↑ V)
        USE(↑ALT↑)                                                 :(A8B)
A12     CC        = POP()
        CC        LEN(1) REM $ CC
A12A    CALL(↑LES↑)
        PUTOUT(SP10 ↑SB1          ↑ CC)
        CALL(↑CNC↑)                                                :(A11A)
A13     OP        = POP()
        CALL($OP)
A13LOOP CC        = POP()                                          :F(A11A)
        PUTOUT(BX1X6)
```

```
553             CC        ↑C↑ =                              :S(A12A)
554             CALL($CC)                                    :(A13LOOP)
     *
     *                    PARAM FUNCTION
     *
555 PARAM       C         SPB =
556             PARAM     = 1
557 PALOOP      EXPRESSION()
558             C         ↑,↑ =                              :S(COMMA)
559             C         END =                              :F(FUNCERR)S(RETURN)
560 COMMA       CALL(↑LES↑)
561             PARAM     = PARAM + 1
562             C         SPB =                              :(PALOOP)
     *
     *                    TOP, POP, PUSH FUNCTIONS
     *
563 TOP         TOP       = 1
564             STACK     BREAK(↑,↑) . T                     :F(RETURN)
565             T         ↑C↑                                :S(TOP2)
566             T         ANY(↑+-↑)                          :S(TOP3)
567             T         ↑V↑                                :S(TOP5)
568             TOP       = 4                                :(RETURN)
569 TOP2        TOP       = 2                                :(RETURN)
570 TOP3        TOP       = 3                                :(RETURN)
571 TOP5        TOP       = 5                                :(RETURN)
572 PUSH        STACK     = INP ↑,↑ STACK                    :(RETURN)
573 POP         STACK     BREAK(↑,↑) $ POP LEN(1) =          :F(FRETURN)S(RETURN)
     *
     *                    CALL, CALL1 FUNCTIONS
     *
574 CALL        PUTOUT(≠  RJ =XFAS$≠ ROU)                    :(RETURN)
575 CALL1       PUTOUT(≠  RJ =XFAS$≠ ROU)
576             PUTOUT(BX1X6)                                :(RETURN)
     *
     *                    LABEL FUNCTION
     *
577 LABEL       DIFFER(LAB)                                  :F(LABEL5)
578             IDENT($(LAB ↑$L1↑))                          :F(DUPLAB)
579             CINDEX    = 1
580             $(LAB ↑L$3↑)  = CONSTANT(LAB)
581             LABEL     = DIFFER($(LAB ↑$L2↑)) $(LAB ↑$L2↑) :S(LABEL4)
582             LABEL     = DIFFER(NEXTLABL) NEXTLABL        :F(LABEL2)
583             NEXTLABL  =
584 LABEL1      LABNUM    = LABNUM + 1
585             $(↑L↑ LABNUM) = LAB
586             $(LAB ↑$L1↑)  = LABEL
587             $(LAB ↑$L2↑) = LABEL                         :(RETURN)
588 LABEL2      LABEL     = NEWLAB()                         :(LABEL1)
589 LABEL4      $(LAB ↑$L1↑) = LABEL
590             DIFFER(NEXTLABL) PUTOUT(NEXTLABL ↑     BSS     0↑)
591             NEXTLABL  =                                  :(RETURN)
592 LABEL5      LABEL     = DIFFER(NEXTLABL) NEXTLABL        :F(FRETURN)
593             NEXTLABL  =                                  :(RETURN)
     *
     *                    ENDBLNM FUNCTION
     *
```

```
594 FNCBLNM   FNBNUM      = FNBNUM + 1
595           FNCBLNM     = FNBNUM                                :(RETURN)
    *
    *                     PDF FUNCTION
    *
596 PDF       PDF         = DIFFER($(PAR '$F')) $(PAR '$F')   :S(RETURN)
597           FNCNUM      = IDENT($(PAR '$FD')) FNCNUM + 1    :F(FDFNCONF)
598           $('F' FNCNUM) = PAR
599           $(PAR '$F') = 'F' FNCNUM
600           PDF         = 'F' FNCNUM                            :(RETURN)
    *
    *                     VAR FUNCTION
    *
601 VAR       VA          = .$(PAR '$V')
602           DIFFER($VA)                                         :F(VAR2)
603           VD          = VR[$VA]
604           IDENT(TYP(VD))                                      :F($TYP(VD))
605 VAR1      VL          = 'V' $VA
606 VAR1.5    LE(REG,5)                                           :F(VOUTR)
607           PUTOUT(SP10 'SA' REG SP7 VL)                        :(RETURN)
608 VOUTR     PUTOUT(SP10 'SA1         ' VL)                      :(FIN1)
609 VAR2      VARNUM      = VARNUM + 1
610           CINDEX      = 1
611           VR[VARNUM]  = VARID(CONSTANT(PAR),)                 :F(VAR2MNY)
612           $VA         = VARNUM                                :(VAR1)
613 VAR3      PATVAR()
614           PVI         = 1
615           PUTOUT('R' PATNUM JP1 'PTF')
616           CALL('REM')
617           PUTOUT(PTS)
618           USE('*')                                            :(VAR1.5)
619 VAR4      PATVAR()
620           PUTOUT('R' PATNUM ' JP =XFAS$PTF')
621           PUTOUT('  JP =XFAS$PTF')
622 VARPF     PUTOUT('  USE *')                                   :(VAR1.5)
623 VAR5      PATVAR()
624           PUTOUT('R' PATNUM ' JP =XFAS$MTA')
625           PUTOUT(PTS)                                         :(VARPF)
626 VAR6      PATVAR()
627           PUTOUT('R' PATNUM ' JP =XFAS$MTA')
628           PUTOUT('  JP =XFAS$MTA')                            :(VARPF)
629 VAR7      PATVAR()
630           PUTOUT('R' PATNUM ' SB2 *+2')
631           PUTOUT('  JP =XFAS$ABR')
632           PUTOUT('  SB2 *+1')
633           PUTOUT('  JP =XFAS$ABS')
634           PUTOUT(PTF)
635           PUTOUT(PTS)                                         :(VARPF)
636 VAR8      PATVAR()
637           PUTOUT('R' PATNUM RJ 'UCR')
638           CALL('BAL')
639           PUTOUT(PTF)
640           CALL('LCR')
641 VAR8.1    PUTOUT('  SB3 B0')
642           PUTOUT(PTS)                                         :(VARPF)
643 VAR9      PUTOUT('R' PATNUM ' JP FS' PATNUM)
```

```
644                 CALL(↑LCR↑)
645                 PUTOUT(PTS)
646                 PUTOUT(↑  USE *↑)
647                 PUTOUT(↑  USE PAT↑)
648                 PUTOUT(↑RS↑ PATNUM ↑RJ =XFAS$RCR↑)        :(VAR8.1)
     *
     *                        VARNM FUNCTION
     *
649 VARNM           VA          = .$(PAR ↑$V↑)
650                 DIFFER($VA)                               :F(VN2)
651                 VD          = VR[$VA]
652                 IDENT(TYP(VD))                            :F(SYNTAX)
653 VN1             VARNM       = ↑V↑ $VA                     :(RETURN)
654 VN2             VARNUM      = VARNUM + 1
655                 CINDEX      = 1
656                 VR[VARNUM]   = VARID(CONSTANT(PAR),)      :F(VAR2MNY)
657                 $VA         = VARNUM                      :(VN1)
     *
     *                        PATVAR FUNCTION
     *
658 PATVAR          PATNUM      = PATNUM + 1
659                 VL          = ↑P↑ PATNUM
660                 PUTOUT(↑  USE DATA↑)
661                 PUTOUT(↑D↑ PATNUM ↑    VFD↑ SP7 ↑30/0,30/R↑ PATNUM)
662                 USE(↑*↑)
663                 USE(↑PAT↑)
664                 PUTOUT(↑P↑ PATNUM ↑    VFD↑ SP7 ↑4/5,56/D↑ PATNUM)
665                 USE(↑*↑)
666                 USE(↑SUB↑)                                :(RETURN)
     *
     *                        CONSTANT FUNCTION
     *
667 CONSTANT        CONS        = .$(CON ↑$C↑ CINDEX)
668                 CONSTANT    = DIFFER($CONS) $CONS         :S(RETURN)
669                 CONNUM      = CONNUM + 1
670                 CN[CONNUM,1] = CON                        :F(CON2MNY)
671                 CN[CONNUM,2]  = CINDEX
672                 $CONS       = ↑C↑ CONNUM
673                 CONSTANT    = ↑C↑ CONNUM                  :(RETURN)
     *
     *                        FUNC FUNCTION
     *
674 FUNC            DIFFER($(FUN ≠$F2≠))                      :F(PRDEF)
675                 GT($(FUN ↑$F2↑),4)                  :S($(↑F.↑ $(FUN ↑$F2↑)))
676                 F           = $(FUN ≠$F2≠)
677 FUNC2           X           = PARAM(≠)≠)
678                 EQ(X,F)                                   :S(FUNCOUT)
679                 GT(X,F)                                   :S(SYNTAX)
680                 EQ(F,1)                                   :S(FUNCOUT)
681                 F           = F - X
682 NULLES          CALL(↑LES↑)
683                 PUTOUT(SP10 ↑BX1        X1-X1↑)
684                 F           = GT(F,1) F - 1               :S(NULLES)
685 FUNCOUT         CALL($(FUN ↑$F1↑))                        :(FIN6)
     *
     *              PROG DEFINED FUNC CALL
```

*

```
686 PRDEF      IDENT($(FUN '$FD'))                                 :F(FDREF)
687            PUTOUT(SP10 'SX2          ' PARAM(')'))
688            PUTOUT('+        SB2        *+1')
689            PUTOUT(SP10 'JP          ' PDF(FUN))                :(FIN6)
690 FDREF      EQ(PARAM(')'),1)                                    :F(FDREFERR)
691            PUTOUT(SP10 'SB1         FD' 10000 + $(FUN '$FD'))
692            CALL('FDT')                                         :(NAM6)
693 F.10       USE('FNCBLK')
694            FNO         = FNCBLNM()
695            PUTOUT('FC' FNO ' SB1 FB' FNO)
696            PUTOUT(JP 'CFN')
697            NARG        = 0
698            NLV         = 1
699            C           (≠'≠ v '≠') $ DEL =                     :F(DEFBAD)
700            C           IDENPT $ FNAME =                        :F(DEFBAD)
701            C           '(' =                                   :F(DEFBAD)
702            C           IDENPT . ARG =                          :F(LV)
703 ARGLOOP    PUTOUT(SP10 'VFD          60/' VARNM(ARG))
704            NARG        = NARG + 1
705            C           ',' =                                   :F(LV)
706            C           IDENPT . ARG =                          :S(ARGLOOP)F(DEFBAD)
707 LV         C.          ')' =                                   :F(DEFBAD)
708            C           IDENPT . LVR =                          :F(STLAB)
709 LVLOOP     PUTOUT(SP10 'VFD          60/' VARNM(LVR))
710            NLV         = NLV + 1
711            C           ',' =                                   :F(STLAB)
712            C           IDENPT . LVR =                          :S(LVLOOP)F(DEFBAD)
713 STLAB      C           DEL =                                   :F(DEFBAD)
714            C           SPB =
715            STLB        = FNAME
716            C           ',' =                                   :F(DEFEND)
717            C           SPB =
718            C           ('≠' v ≠'≠) $ DEL IDENPT $ STLB  DEL =  :F(DEFBAD)
719 DEFEND     C           SPB =
720            C           ')' =                                   :F(DEFBAD)
721            PUTOUT(SP10 'VFD          60/' VARNM(FNAME))
722            NTOT        = NARG + NLV
723            PUTOUT('FB' FNO '    VFD          12/' NTOT ',18/PLIST,12/' NTOT
723     ',18/0')
724            PUTOUT(SP10 'VFD          18/' NARG ',24/' NLV ',18/0')
725            USE('*')
726            PUTOUT('  SX2 FC' FNO)
727            PUTOUT(SP10 'SX1          ' GOTOP(STLB))
728            PUTOUT(SP10 'SB1          ' PDF(FNAME))
729            CALL('DFN')                                         :(RETURN)
730 F.5        LE(PARAM(')'),1)                                    :F(PAR2MNY)
731            CALL('MSI')
732            PUTOUT(SP10 'NG' SP8 'X6,' EX 'ERR+2')
733            PATNUM      = PATNUM + 1
734            PUTOUT(SP10 'SA6          D' PATNUM '+1')
735            PUTOUT(SP10 'SA1          P' PATNUM)
736            USE('DATA')
737            PUTOUT('D' PATNUM '    VFD          30/1,30/R' PATNUM)
738            PUTOUT(SP10 'BSSZ         1')
739            USE('*')
```

```
740           USE(*PAT*)
741           PUTOUT(*P* PATNUM *    VFD        4/5,56/D* PATNUM)
742           USE(***)
743           USE(*SUB*)
744           PUTOUT(*R* PATNUM JP1 *PTF*)
745           PUTOUT(SP10 *SA1       A3+1*)
746           IDENT(FUN,*POS*)                               :S(F.6)
747           IDENT(FUN,≠≠POS≠)                              :S(F.7)
748           CALL($(FUN *$F1*))
749           PUTOUT(PTF)
750           PUTOUT(PTS)
751           USE(***)                                       :(FIN1)
752 F.6       PUTOUT(≠  SA2 ≠ EX ≠SBJ+1≠)
753           PUTOUT(≠  AX2 48≠)
754           PUTOUT(≠  IX1 X2-X1≠)
755 F.7       PUTOUT(≠  SB1 X1≠)
756           PUTOUT(≠  EQ B1,B4,≠ EX ≠PTS≠)
757           PUTOUT(JP ≠PTF≠)
758           PUTOUT(USEAST)                                 :(FIN1)
759 F.8       LE(PARAM(*)*),1)                               :F(PAR2MNY)
760           BTABNO     = BTABNO + 1
761           PUTOUT(SP10 *SB1        K* BTABNO)
762           CALL(*BKT*)
763           USE(*BREAK*)
764           PUTOUT(≠K≠ BTABNO ≠ BSSZ 2≠)
765           USE(***)
766 F.85      PATNUM     = PATNUM + 1
767           PUTOUT(SP10 *SA6         D* PATNUM *+1*)
768           PUTOUT(SP10 *SA1         P* PAT IUM)
769           USE(*DATA*)
770           PUTOUT(*D* PATNUM *    VFD        30/1,30/R* PATNUM)
771           PUTOUT(SP10 *BSSZ        1*)
772           USE(***)
773           USE(*PAT*)
774           PUTOUT(*P* PATNUM *    VFD        4/5,56/D* PATNUM)
775           USE(***)
776           USE(*SUB*)
777           PUTOUT(*R* PATNUM JP1 *PTF*)
778           PUTOUT(SP10 *SA1       A3+1*)
779           CALL($(FUN *$F1*))
780           PUTOUT(PTF)
781           PUTOUT(PTS)
782           USE(***)                                       :(FIN1)
783 F.9       C          ANY(*≠* *≠≠) $ DEL IDENPT $ NAM *(* =  :F(SYNTAX)
784           TYPENO     = LE(TYPENO,510) TYPENO + 1     :F(TYP2MNY)
785           PUTOUT(SP10 *SB1         * PDF(NAM))
786           PUTOUT(SP10 *SX2         T* 10000 + TYPENO)
787           CALL(*DDT*)
788           USE(*DATA*)
789           PUTOUT(*T* 10000 + TYPENO *    SB1          *+1*)
790           PUTOUT(JP *GOT*)
791           PUTOUT(SP10 *VFD         4/6,10/* TYPENO *,10/NF* TYPENO
791 .         *,36/NF* TYPENO)
792           CINDEX     = 1
793           FXNO       = FXNO + 1
794           FX[FXNO]   = *30/* CONSTANT(NAM) *,30/T* 10000 + TYPENO *+1*
```

```
795              NF          = 0
796 F.9LOF       C           IDENPT $ F1 NSPAN(* *) *,* NSPAN(* *) IDENPT $ F2
796 .            NSPAN(* *) =                                 :F(PARTIAL)
797              NF          = NF + 2
798              PUTOUT(SP10 *VFD        30/* FIELD(F1) *,30/* FIELD(F2))
799              C             *,* NSPAN(* *) =               :S(F.9LOF)
800              C             *)* NSP DEL NSP *)* =          :F(DATABAD)
801 F.9FIN       PUTOUT(*NF* TYPEND SP7 *EQU           * NF)
802              USE(***)                                     :(FIN1)
803 PARTIAL      C           IDENPT $ F1 NSP *)* NSP DEL NSP *)* =   :F(DATABAD)
804              PUTOUT(SP10 *VFD        30/* FIELD(F1) *,30/0*)
805              NF          = NF + 1                         :(F.9FIN)
806 F.11         PARAM(*)*)                                   :(RETURN)
807 F.12         LE(PARAM(*)*),1)                             :F(PAR2MNY)
808              PUTOUT(BX6X1)
809              PUTOUT(*   SA6 =XFAS$COM+8*)                 :(RETURN)
810 F.13         PUTOUT(*   SB1 4*)
811              PUTOUT(*   RJ =XFAS$FKW*)                    :(FIN6)
812 F.14         PUTOUT(*   SB1 * $(FUN *$F2*))
813              PUTOUT(*   RJ =XFAS$KFN*)                    :(FIN6)
814 F.15         LE(PARAM(*)*),1)                             :F(PAR2MNY)
815              PUTOUT(BX6X1)
816              PATNUM      = PATNUM + 1
817              PUTOUT(* SA6 D* PATNUM *+1*)
818              PUTOUT(*   SA1 P* PATNUM)
819              PUTOUT(*   USE DATA*)
820              PUTOUT(*D* PATNUM * VFD 30/1,30/R* PATNUM)
821              PUTOUT(*   BSSZ 1*)
822              PUTOUT(*   USE PAT*)
823              PUTOUT(*P* PATNUM * VFD 4/5,56/D* PATNUM)
824              PUTOUT(*   USE SUB*)
825              PUTOUT(*F* PATNUM * JP RS* PATNUM)
826              PUTOUT(*   SX1 =XFAS$PTF*)
827              PUTOUT(*   RJ =XFAS$LES*)
828              PUTOUT(*AS* PATNUM * RJ =XFAS$LCR*)
829              PUTOUT(*   JP =XFAS$PTS*)
830              PUTOUT(*RS* PATNUM * RJ =XFAS$UCR*)
831              PUTOUT(*   ZR B3,*+3*)
832              PUTOUT(*AF* PATNUM * RJ =XFAS$UES*)
833              PUTOUT(*   SB1 X6*)
834              PUTOUT(*   JP B1*)
835              PUTOUT(*   SA1 D* PATNUM *+1*)
836              PUTOUT(*   RJ =XFAS$PTP*)
837              PUTOUT(*   NZ B1,AF* PATNUM)
838              PUTOUT(*PS* PATNUM * SX1 PR* PATNUM)
839              PUTOUT(*   SB3 B0*)
840              PUTOUT(*   RJ =XFAS$LES*)
841              PUTOUT(*   JP AS* PATNU)
842              PUTOUT(*PR* PATNUM * SA1 D* PATNUM *+1*)
843              PUTOUT(*   RJ =XFAS$PRT*)
844              PUTOUT(*   NZ B1,PF* PATNUM)
845              PUTOUT(*   JP PS* PATNUM)
846              PUTOUT(*   USE*)                             :(FIN1)
847 F.16         PUTOUT(*   SB1 7*)
848              PUTOUT(*   RJ =XFAS$FKW*)                    :(FIN6)
849 F.17         LE(PARAM(*)*),1)                             :F(PAR2MNY)
```

```
850                PUTOUT(BX6X1)                                          :(F.85)
851 F.13           P          = PARAM(≠)≠)
852                LE(P,3)                                                :F(PAR2MNY)
853 F.131          P          = LT(P,3) P + 1                             :F(F.182)
854                PUTOUT(≠  RJ =XFAS$LES≠)
855                PUTOUT(≠  BX1 X1-X1≠)                                  :(F.181)
856 F.182          BTABNO     = BTABNO + 1
857                PUTOUT(≠  SX6 K≠ BTABNO)
858                PUTOUT(≠  USE BREAK≠)
859                PUTOUT(≠K≠ BTABNO ≠ BSSZ 3≠)
860                PUTOUT(≠  USE≠)
861                PUTOUT(≠  RJ =XFAS$RPL≠)                               :(FIN6)
    *
    *                         FIELD FUNCTION
    *
862 FIELD          IDENT($(FD ↑$F↑))                                      :F(FDFNCONF)
863                FDN        = $(FD ↑$FD↑)
864                FDNUM      = IDENT(FDN) FDNUM + 1                      :S(NEWFD)
865                F          = FLD[FDN]
866                TYPENO(F)  = TYPENO(F) + 1
867                LAST(F)    = FDATA(TYPENO,LAST(F))
868                FIELD      = ↑FD↑ 10000 + FDN                          :(RETURN)
869 NEWFD          $(FD ↑$FD↑) = FDNUM
870                FLD[FDNUM] = FDATA(1,FDATA(TYPENO,0))                  :F(FD2MNY)
871                FIELD      = ↑FD↑ 10000 + FDNUM                        :(RETURN)
    *
    *                         FUNCNAME FUNCTION
    *
872 FUNCNAME       IDENT($(FUN ↑$FD↑))                                    :F(FDNAME)
873                FUNC(FUN,REG)                                          :(RETURN)
874 FDNAME         EQ(PARAM(↑)↑),1)                                       :F(FDREFERR)
875                PUTOUT(SP10 ↑SB1          FD↑ 10000 + $(FUN ↑$FD↑))
876                CALL(↑FDT↑)                                            :(FIN6)
    *
    *                         PUTOUT FUNCTION
    *
877 PUTOUT         PUT        = DIFFER(ASMFLG) PAR
878                OUTPUT     = DIFFER(LISTOB) INDENT PAR                 :(RETURN)
    *
    *                         ERROR  MESSAGES
    *
879 SYNTAX         ERRMSG(↑SYNTAX ERROR IN STATEMENT↑)                    :(STLOOP)
880 FEATURE        ERRMSG(↑FEATURE NOT YET IMPLEMENTED↑)                  :(STLOOP)
881 MISQUOTE       ERRMSG(↑MISSING RIGHT QUOTE↑)                          :(STLOOP)
882 DUPLAB         ERRMSG(↑DUPLICATE LABEL IGNORED↑)                      :(STLOOP)
883 BADGOT         ERRMSG(↑SYNTAX ERROR IN GOTO PART↑)                    :(STLOOP)
884 NONAME         ERRMSG(↑NAME EXPECTED, NONE FOUND↑)                    :(STLOOP)
885 SYSERR         ERRMSG(≠AN INTERNAL COMPILER ERROR HAS BEEN DETECTED. PLEASE R
885 .EPORT THIS PROBLEM.≠)                                                :(ENDMSG)
886 UNBAL          ERRMSG(≠UNBALANCED PARENTHESES≠)                       :(STLOOP)
887 KEYERR         ERRMSG(≠ILLEGAL OR MISSPELLED KEYWORD≠)                :(STLOOP)
888 NORQOT         ERRMSG(≠MISSING QUOTATION MARK≠)                       :(STLOOP)
889 MISPAR         ERRMSG(≠MISSING RIGHT PARENTHESES≠)                    :(STLOOP)
890 FUNCERR        ERRMSG(≠MISSING ≠ END )                                :(STLOOP)
891 FDFNCONF       ERRMSG(≠FIELD AND FUNCTION NAMES CONFLICT≠)                :(STLOOP)
892 VAR2MNY        ERRMSG(≠INTERNAL TABLE OVERFLOW/ TOO MANY VARIABLES≠) :(STLOOP
```

```
892 .)
893 CON2MNY   ERRMSG(≠INTERNAL TABLE OVERFLOW/ TOO MANY CONSTANTS≠) :(STLOOF
893 .)
894 FDREFERR  ERRMSG(≠MORE THAN ONE PARAMETER TO A FIELD REFERENCE≠)
894 .                                                    :(STLOOP)
895 DEFBAD    ERRMSG(≠SYNTAX ERROR IN A DEFINE FUNCTION≠)   :(STLOOP)
896 PAR2MNY   ERRMSG(≠TOO MANY PARAMETERS TO A BUILT-IN FUNCTION≠)
896 .                                                    :(STLOOP)
897 TYP2MNY   ERRMSG(≠INTERNAL TABLE OVERFLOW/ TOO MANY PROGRAMMER-DEFINED D
897 .ATATYPES≠)                                          :(STLOOP)
898 DATABAD   ERRMSG(≠SYNTAX ERROR IN A DATA FUNCTION≠)     :(STLOOP)
899 FD2MNY    ERRMSG(≠INTERNAL TABLE OVERFLOW/ TOO MANY FIELDS≠) :(STLOOP)
900 FX2MNY    ERRMSG(≠INTERNAL TABLE OVERFLOW/ TOO MANY CONSTANTS OVER 8 CHA
900 .RACTERS LONG≠)                                      :(STLOOP)
901 ERRMSG    OUTPUT = ↑ *****   ↑ PAR ↑   *****↑
902           DIAGNO    = DIAGNO + 1                     :(RETURN)
    *
    *                   GETSA FUNCTION
    *
903 GETSA     EQ(MSLI)                                   :F(GETSA2)
904 GETSA1    CURLIN    = DIFFER(NXTLIN) NXTLIN          :F(FRETURN)
905           LINNUM    = LINNUM + 1
906           NXTLIN    = GETLIN()                       :S(GETSA2)
907           NXTLIN    =
    *                   CHECK FOR COMMENT OR CONTROL
908 GETSA2    CURLIN    ANY(↑-*↑) $ STR1                 :F(GETSA5)
909           (EQ(MSLI) PUTOUT(↑* ↑ CURLIN) NE(LISTSR) PRTLIN(LPAD(LINNUM,6
909 .   ,↑0↑) INDENT INDENT CURLIN))
    *                   COMMENT
910           DIFFER(STR1,↑*↑)                           :F(GETSA1)
    *                   CONTROL
911           INTGER    =
912           CURLIN    LEN(1) NSPAN(↑ ↑) SPAN(↑ABCDEFGHIJKLMNOPQRSTUVWXYZ↑
912 .   ) $ STR1 NSPAN(↑ ↑) FENCE (RPOS(0) v INTGPT NSPAN(↑ ↑) RPOS(0))
912 .                                                    :F(GETSA4)
913           STR1      = $(STR1 ↑.↑)
914           IDENT(STR1)                   :F($STR1)
915 GETSA4    ERRMSG(↑BAD CONTROL LINE↑)                 :(GETSA1)
    *                   LIST
916 C.1       LISTSR    = 1                              :(GETSA1)
    *                   UNLIST
917 C.2       LISTSR    = 0                              :(GETSA1)
    *                   NOCODE
918 C.3       LISTOB    =                                :(GETSA1)
    *                   CODE
919 C.4       LISTOB    = 1                              :(GETSA1)
    *                   EJECT
920 C.5       EJECT()                                    :(GETSA1)
    *                   FAIL
921 C.6       NOFAIL    =                                :(GETSA1)
    *                   NOFAIL
922 C.7       NOFAIL    = ↑=XFAS$ERR+11↑                 :(GETSA1)
    *                   NOCROSS
923 C.8       CRSFLG    = 0                              :(GETSA1)
    *                   CROSSREF
924 C.9       CRSFLG    = 1
```

```
925              CRSIND     = 1                                  :(GETSA1)
     *                      NOASM
926 C.10         ASMFLG     = 1                                  :(GETSA1)
     *                      LISTASM
927 C.11         ASMFLG      = 2                                 :(GETSA1)
     *                      SPACE N
928 C.12         INTGER     = IDENT(INTGER) 1
929              SPCLIN(INTGER)                                  :(GETSA1)
     *                      NEWSTNO N
930 C.13         NE(INTGER)                                      :F(GETSA4)
931              MAXSTN     = GT(STNO,MAXSTN) STNO
932              STNO       = INTGER - 1                         :(GETSA1)
933 C.14         INTGER     = IDENT(INTGER) 0
934              PUTOUT(≠  SX6≠ INTGER)
935              PUTOUT(SA6 ≠SNO≠)                               :(GETSA1)
     *                      CSNAP N
936 C.15         CCMSNAP(INTGER)                                 :(GETSA1)
     *                      NEXT STATEMENT
937 GETSA5       STNO       = STNO + 1
938              (EQ(MSLI) PUTOUT(↑* ↑ CURLIN) NE(LISTSR) PRTLIN(
938 .  LPAD(LINNUM,6,↑0↑) INDENT LPAD(STNO,5,↑ ↑) ↑ ↑ CURLIN))
939 GETSA6       CURLIN     REALCH(↑;↑)                          :F(GETSA7)
940              (GETSA CURLIN) SEMIPT $ STR1                    :F(GETSA7)
     *                      MULTI-STATEMENT LINE
941              MSLI       = 1
942              CURLIN     STR1 ↑;↑ =
943              GETSA      = PUTOUT(↑* ↑) GETSA STR1            :(RETURN)
944 GETSA7       GETSA      = GETSA CURLIN
     *           CONTINUATION CHECK
945              NXTLIN     ANY(↑.+↑) REM $ CURLIN               :F(GETSA8)
946              LINNUM     = LINNUM + 1
947              (PUTOUT(↑* ↑ NXTLIN) NE(LISTSR) PRTLIN(LPAD(LINNUM,6,↑0↑)
947 .  INDENT LPAD(STNO,5,↑ ↑) ↑ ↑ NXTLIN))
948              NXTLIN     = GETLIN()                           :S(GETSA6)
949              NXTLIN     =                                    :(GETSA6)
950 GETSA8       MSLI       = PUTOUT(↑* ↑) 0                     :(RETURN)
     *
     *                      EACTPH FUNCTION
     *
951 ENDA
952 EACTPH       BNUM       = 10000
953              P          = EXNO
954              PUTOUT(JP ↑END↑)
955              N          = 10000
956 CLOOP        N          = LT(N,CONNUM) N + 1                 :F(CONOUT)
957              A          = CN[N,1]                            :($(↑CL↑ CN[N,2]))
958 CL2          PUTOUT(↑C↑ N ↑     VFD       4/2,56/↑ A)  :(CLOOP)
959 CL3          PUTOUT(↑C↑ N ↑ VFD 4/1,56/EL↑ N)
960              PUTOUT(↑RL↑ N ↑ DATA ↑ A)                       :(CLOOP)
961 CL1          M          = SIZE(A)
962 CONOTNO      GT(M,8)                                         :S(CONSTR)
963              PUTOUT(↑C↑ N ↑     VFD       12/↑ M ↑,48/↑ M ↑L↑ A) :(CLOOP)
964 CONSTR       I          = WORDS(M)
965              BNUM       = BNUM + 1
966              PUTOUT(↑C↑ N ↑     VFD       4/8,20/↑ M ↑,18/0,18/B↑ BNUM)
967              P          = P + 1
```

```
968            PUTOUT(≠  VFD 1/1,23/≠ M - (I - 1) * 10 ≠,18/≠ I
968 .  ↑,18/1↑)
969            GT(M,50)                                          :F(LE50)
970            A          LEN(50) $ X REM $ A
971            PUTOUT(≠B≠ BNUM ≠ DATA 50L≠ X)
972            M          = M - 50
973 GTLP       A          LEN(60) $ X REM $ A                    :F(GTLS)
974            M          = M - 60
975            PUTOUT(≠  DATA 60L≠ X)                            :(GTLP)
976 GTLS       IF(DIFFER(A) PUTOUT(≠  DATA ≠ M ≠L≠ A))           :(GTSK)
977 LE50       PUTOUT(↑B↑ BNUM ↑ DATA ↑ M ↑L↑ A)
978 GTSK
979            FX[P]      = ↑30/B↑ BNUM ↑+↑ I - 1 ↑,30/C↑ N :S(CLOOP)F(FX2MNY)
980 CONOUT     N          = 10002
981 VLOOP      N          = LT(N,VARNUM) N + 1                   :F(VAROUT)
982            PUTOUT(≠V≠ N ≠ VFD 4/3,56/*+1≠)
983            PUTOUT(≠  BSSZ 1≠)                                :(VLOOP)
984 VAROUT     PUTOUT(↑V10001     VFD       4/3,18/2,38/Z↑ N + 1)
985            PUTOUT(↑V10002     VFD       4/3,18/1,38/Z↑ N + 2)
986            PUTOUT(↑Z↑ (N + 1) ↑     DATA       0↑)
987            PUTOUT(↑Z↑ (N + 2) ↑     DATA       0↑)
988            FX[P + 1]  = ↑30/FAS$CAP,30/V10001↑
989            FX[P + 2]  = ↑30/FAS$IAP,30/V10002↑
989 .                                                            :F(FX2MNY)
990            PUTOUT(SP10 ↑EXT        FAS$OFT,FAS$IFT↑)
991            P          = P + 2
992            N          = 10000
993 FNCLOOP    N          = LT(N,FNCNUM) N + 1                   :F(FNCOUT)
994            PUTOUT(≠F≠ N ≠ JP =XFAS$ERR+4≠)                   :(FNCLOOP)
995 FNCOUT     N          = 10000
996 FDLOOP     N          = LT(N - 10000,FDNUM) N + 1  :F(FDOUT)
997            F          = FLD[N - 10000]
998            M          = TYPENO(F)
999            PUTOUT(↑FD↑ N ↑     DATA        ↑ M)
1000           I          = 0
1001 TYPLOPE    I          = LT(I,M) I + 1                        :F(FDLOOP)
1002           F          = LAST(F)
1003           PUTOUT(SP10 ↑DATA        ↑ TYPENO(F))          :(TYPLOPE)
1004 FDOUT      PUTOUT(↑PLIST     DATA        ↑ SIZE(PROG) ↑L↑ PROG)
1005           PUTOUT(SP10 ↑DATA       0↑)
1006           PUTOUT(SP10 ↑DATA       ↑ P)
1007           N          = 0
1008 PLOOP      N          = LT(N,P) N + 1                        :F(POUT)
1009           PUTOUT(SP10 ↑VFD         ↑ FX[N])              :(PLOOP)
1010 POUT       N          = 10000
1011           PUTOUT(SP10 ↑DATA        ↑ VARNUM - 10000)
1012 ILOOP      N          = LT(N,VARNUM) N + 1                   :F(IOUT)
1013           PUTOUT(SP10 ↑VFD        30/↑ CON(VR[N]) ↑,30/V↑ N)  :(ILOOP)
1014 IOUT       N          = 10000
1015           PUTOUT(SP10 ↑DATA        ↑ LABNUM - N)
1016 LLOOP      N          = LT(N,LABNUM) N + 1                   :F(LOUT)
1017           LAB        = $(↑L↑ N)
1018           IF(IDENT($(LAB ↑$L1↑)) PUTOUT($(LAB ↑$L2↑) ↑   EQU        FAS$
1018 .ERR+3↑))                                                   :F(LA2)
1019           ERRMSG(LAB ↑ IS AN UNDEFINED LABEL↑)
1020 LA2        PUTOUT(≠  USE GOTO≠)
```

```
1021            PUTOUT(≠LL≠ N ≠ VFD 4/3,8/1,48/≠ $(LAB ≠$L2≠))
1022            PUTOUT(USEAST)
1023            PUTOUT(≠  VFD 30/≠ $(LAB ≠L$3≠) ≠,30/LL≠ N)   :(LLOOP)
1024 LOUT       PUTOUT(SP10 ↑END         ↑ PROG)
1025            REWIND(↑ASM↑)
1026            SPCLIN(2)
1027            OUTPUT = VARNUM - 10000 ≠ VARIABLES ENCOUNTERED≠
1028            OUTPUT = (CONNUM - 10000 ≠ CONSTANTS SEEN≠
1029            OUTPUT = P ≠ FIXS MADE≠
1029 .                                                  :(ENDMSG)
     *                          SPCLIN, PRTLIN, GETLIN FUNCTIONS
     *
1030 SPCLIN     N          = PAR
1031 SPCLIN1    N          = GT(N,0) N - 1              :F(RETURN)
1032            OUTPUT     =                            :(SPCLIN1)
1033 PRTLIN     OUTPUT     = PAR                        :(RETURN)
1034 GETLIN     GETLIN     = INPUT                      :S(RETURN)F(FRETURN)
     *
     *                          INISYM FUNCTION
     *
1035 INISYM     CONNUM     = 10000
1036            LABNUM     = 10000
1037            $(↑END$L2↑) = ↑FAS$END↑
1038            $(↑RETURN$L2↑) = ↑FAS$SFN↑
1039            $(↑FRETURN$L2↑) = ↑FAS$FFN↑
1040            $(↑NRETURN$L2↑) = ↑FAS$SFN↑
1041            VARNUM     = 10002
1042            VR         = ARRAY(↑9994/10500↑)
1043            VR[9994]   = VARID(,↑VAR4↑)
1044            VR[9995]   = VARID(,↑VAR5↑)
1045            VR[9996]   = VARID(,↑VAR6↑)
1046            VR[9997]   = VARID(,↑VAR7↑)
1047            VR[9998]   = VARID(,↑VAR8↑)
1048            VR[9999]   = VARID(,↑VAR9↑)
1049            FAIL$V     = 9994
1050            FENCE$V    = 9995
1051            ABORT$V    = 9996
1052            ARB$V      = 9997
1053            BAL$V      = 9998
1054            SUCCEED$V  = 9999
1055            VR[10000]   = VARID(,↑VAR3↑)
1056            $(↑REM$V↑) = 10000
1057            CINDEX     = 1
1058            VR[10001]   = VARID(CONSTANT(↑OUTPUT↑),)
1059            VR[10002]   = VARID(CONSTANT(↑INPUT↑),)
1060            FNCNUM     = 10000
1061            FNBNUM     = 10000
1062            $(↑OUTPUT$V↑)   = 10001
1063            $(↑INPUT$V↑)    = 10002
1064            PATNUM     = 10000
1065            BTABNO     = 10000
1066            ALTP       = 10000
1067            STFCOUNT.  = 1
1068            LASTNO.    = 2
1069            STNO.      = 3
1070            FNCLEVEL.  = 4
```

```
STCOUNT.   = 5
RTNTYPE.   = 6
ALPHABET.  = 7
ABEND.     = 10
ANCHOR.    = 11
FULLSCAN.  = 12
MAXLNGTH.  = 13
STLIMIT.   = 14
SPECIAL.   = 15
FLD        = ARRAY(100)
FNC(↑TRIM↑,↑TRM↑,1)
FNC(↑ARRAY↑,↑GAR↑,2)
FNC(↑IDENT↑,↑IDT↑,2,1)
FNC(↑DIFFER↑,↑DIF↑,2,1)
FNC(↑LT↑,↑LTP↑,2,1)
FNC(↑LE↑,↑LEP↑,2,1)
FNC(↑GT↑,↑GTP↑,2,1)
FNC(↑GE↑,↑GEP↑,2,1)
FNC(↑EQ↑,↑EQP↑,2,1)
FNC(↑NE↑,↑NEP↑,2,1)
FNC(↑SIZE↑,↑SIZ↑,1)
FNC(↑REWIND↑,↑REW↑,1,1)
FNC(↑DEFINE↑,↑DFN↑,10)
FNC(↑LEN↑,↑LEN↑,5)
FNC(↑TAB↑,↑TAB↑,5)
FNC(↑RTAB↑,↑RTB↑,5)
FNC(≠RPOS≠,,5)
FNC(≠POS≠,,5)
FNC(↑SPAN↑,↑SPN↑,8)
FNC(↑BREAK↑,↑BRK↑,8)
FNC(↑ANY↑,↑ANY↑,8)
FNC(↑NOTANY↑,↑NTA↑,8)
FNC(↑NSPAN↑,↑NSP↑,8)
FNC(↑REALCH↑,↑RCH↑,8)
FNC(↑OUTPUT↑,↑OTP↑,4,1)
FNC(↑INPUT↑,↑INP↑,3,1)
FNC(↑LPAD↑,↑LPD↑,3)
FNC(↑RPAD↑,↑RPD↑,3)
FNC(↑IF↑,,11,1)
FNC(↑DATA↑,↑DDT↑,9)
FNC(≠COMSNAP≠,≠CSP≠,1,1)
FNC(≠TIME≠,≠TME≠,1)
FNC(≠CLOCK≠,≠CLK≠,1)
FNC(↑ANCHOR↑,,12,1)
FNC(↑DETACH↑,↑DT↑,1,1)
FNC(↑EOI↑,↑EOI↑,1,1)
FNC(↑ENDGROUP↑,↑EGR↑,2,1)
FNC(↑EOR↑,↑EOR↑,1,1)
FNC(↑FNCLEVEL↑,,13)
FNC(↑MAXLNGTH↑,11,14)
FNC(↑STCOUNT↑,5,14)
FNC(↑STLIMIT↑,12,14)
FNC(↑ARBNO↑,,15)
FNC(↑ALPHABET↑,,16)
FNC(↑REALSTR↑,↑RLS↑,17)
FNC(↑BREAKSTR↑,↑BST↑,17)
```

```
1127               FNC(≠REPLACE≠,≠RPL≠,18)
1128               FNC(≠SUBSTR≠,≠SBS≠,3)
1129               FNC(↑DATE↑,↑DTE↑,1)                                :(RETURN)
     *
     *                          FNC FUNCTION
     *
1130 FNC           $(P1  ↑$F1↑)  =  P2
1131               $(P1  ↑$F3↑)   =  P4
1132               $(P1  ↑$F2↑)  =  P3                                :(RETURN)
     *
     *                          EJECT, GOBBLE FUNCTIONS
     *
1133 EJECT         PAGE  =                                            :(RETURN)
     *
     *                          WORDS, USE FUNCTIONS
     *
1134 WORDS         I             = P / 10
1135               J             = P - (I * 10)
1136               I             = NE(J,0) I + 1
1137               WORDS         = I                                  :(RETURN)
1138 USE           PUTOUT(SP10 ↑USE         ↑ STR)                    :(RETURN)
     *
     *                          NOEND  I.E. NO END CARD
     *
1139 NOEND         DIFFER(NEXTLABL)                                   :F(ENDA)
1140               PUTOUT(NEXTLABL ↑  EQU FAS$END↑)                   :(ENDA)
```

APPENDIX II

List of Runtime Routines

FLIB1.2C PROCESSING LIBRARY FROM BIN AT 12:45:53 30 AUG 73
OPTIONS; COMMON MAP

PROGRAM	LENGTH	INDEX	DATE	TIME	PROCESSOR
FAS$SBS	000065	00000001	26 AUG 73	14:07:04	CMP2.4A0
FAS$INP	000054	00000003	26 AUG 73	14:38:35	CMP2.4A0
FAS$REW	000021	00000005	12 AUG 73	17:14:42	CMP2.4A0
FAS$_PD	000024	00000006	12 AUG 73	22:08:37	CMP2.4A0
FAS$BAL	000015	00000007	25 JUN 73	00:55:32	CMP2.4A0
FAS$CSP	000003	00000010	25 JUN 73	00:55:33	CMP2.4A0
FAS$RPL	000111	00000011	18 AUG 73	23:25:09	CMP2.4A0
FAS$SBR	001015	00000013	24 JUN 73	23:04:41	CMP2.4A0
FAS$MCM	000004	00000014	24 JUN 73	23:04:20	CMP2.4A0
FAS$CLK	000013	00000015	31 MAY 73	19:53:19	CMP2.4A0
FAS$TME	000016	00000016	31 MAY 73	19:53:21	CMP2.4A0
FAS$DTP	000114	00000017	19 AUG 73	13:25:48	CMP2.4A0
FAS$CSR	000017	00000021	04 JUN 73	22:48:12	CMP2.4A0
FAS$RCH	000025	00000022	24 JUN 73	23:04:34	CMP2.4A0
FAS$PSR	000037	00000023	25 APR 73	16:35:00	CMP2.3A0
FAS$DAT	000076	00000024	24 JUN 73	23:04:01	CMP2.4A0
FAS$ASR	000037	00000026	25 APR 73	16:32:24	CMP2.3A0
FAS$CVR	000037	00000027	25 APR 73	16:32:58	CMP2.3A0
DUMPREG	000011	00000030	12 NOV 70	12:32:32	CMP1.1A4
RESTORE	000042	00000031	12 NOV 70	12:32:32	CMP1.1A4
SAVEREG	000123	00000032	12 NOV 70	12:32:32	CMP1.1A4
FAS$ACV	000037	00000034	24 JUN 73	23:03:44	CMP2.4A0
FAS$ANG	000030	00000036	25 APR 73	16:32:13	CMP2.3A0
FAS$ARB	000013	00000037	24 JUN 73	23:03:46	CMP2.4A0
FAS$ANY	000024	00000040	24 JUN 73	23:03:45	CMP2.4A0
FAS$ARO	000076	00000041	25 APR 73	16:32:18	CMP2.3A0
FAS$ASC	000005	00000043	24 JUN 73	23:03:47	CMP2.4A0
FAS$ASG	000031	00000044	24 JUN 73	23:03:48	CMP2.4A0
FAS$BRK	000011	00000046	24 JUN 73	23:03:51	CMP2.4A0
FAS$BKT	000033	00000047	24 JUN 73	23:03:50	CMP2.4A0
FAS$CFN	000072	00000050	04 AUG 73	16:15:48	CMP2.4A0
FAS$CKA	000006	00000052	25 APR 73	16:32:45	CMP2.3A0
FAS$CNC	000330	00000053	04 AUG 73	16:15:51	CMP2.4A0
FAS$COM	000564	00000057	24 JUN 73	23:03:59	CMP2.4A0
FAS$CPS	000015	00000062	25 APR 73	16:32:55	CMP2.3A0
FAS$DFN	000007	00000063	24 JUN 73	23:04:03	CMP2.4A0
FAS$DID	000003	00000064	25 APR 73	16:33:05	CMP2.3A0
FAS$DRD	000006	00000065	25 APR 73	16:33:05	CMP2.3A0
FAS$DSA	000012	00000066	25 APR 73	16:33:06	CMP2.3A0
FAS$DSD	000021	00000067	30 APR 73	04:42:00	CMP2.3A0
FAS$DTE	000013	00000070	25 APR 73	16:33:08	CMP2.3A0
FAS$END	000015	00000071	31 MAY 73	19:53:26	CMP2.4A0
FAS$ERR	000033	00000072	25 APR 73	16:33:15	CMP2.3A0
FAS$ESR	000042	00000073	25 APR 73	16:33:16	CMP2.3A0
FAS$EXP	000014	00000075	25 APR 73	16:33:18	CMP2.3A0
FAS$FKW	000021	00000076	31 MAY 73	19:53:27	CMP2.4A0
FAS$FLR	000004	00000077	30 APR 73	04:42:02	CMP2.3A0
FAS$FRS	000366	00000100	04 AUG 73	16:15:54	CMP2.4A0
FAS$GAR	000232	00000106	24 JUN 73	23:04:09	CMP2.4A0

FAS$GPB	000007	00000111	24 JUN 73 23:04:12	CMP2.4A0
FAS$IAT	000113	00000112	26 AUG 73 14:38:30	CMP2.4A0
FAS$ILZ	000062	00000114	19 AUG 73 13:25:45	CMP2.4A0
FAS$IOC	000030	00000116	19 AUG 73 13:25:47	CMP2.4A0
FAS$INT	000015	00000120	24 JUN 73 23:04:15	CMP2.4A0
FAS$IPR	000052	00000121	24 JUN 73 23:04:15	CMP2.4A0
FAS$ISG	000003	00000123	25 APR 73 16:33:50	CMP2.3A0
FAS$ITS	000074	00000124	25 APR 73 16:33:51	CMP2.3A0
FAS$IVV	000031	00000126	25 APR 73 16:33:53	CMP2.3A0
FAS$LEN	000005	00000130	24 JUN 73 23:04:16	CMP2.4A0
FAS$LGT	000026	00000131	25 APR 73 16:34:27	CMP2.3A0
FAS$MAT	000102	00000132	24 JUN 73 23:04:17	CMP2.4A0
FAS$MKI	000007	00000135	25 APR 73 16:34:33	CMP2.3A0
FAS$MKN	000015	00000136	25 APR 73 16:34:35	CMP2.3A0
FAS$MKP	000032	00000137	24 JUN 73 23:04:21	CMP2.4A0
FAS$MKS	000007	00000141	25 APR 73 16:34:37	CMP2.3A0
FAS$MSI	000024	00000142	25 APR 73 16:34:38	CMP2.3A0
FAS$MSK	000013	00000143	25 APR 73 16:34:40	CMP2.3A0
FAS$MST	000045	00000144	24 JUN 73 23:04:22	CMP2.4A0
FAS$MTR	000104	00000146	12 AUG 73 17:14:23	CMP2.4A0
FAS$MVS	000036	00000150	26 AUG 73 14:06:44	CMP2.4A0
FAS$OAT	000064	00000151	24 JUN 73 23:04:26	CMP2.4A0
FAS$OUT	000022	00000153	25 APR 73 16:34:53	CMP2.3A0
FAS$PAR	000004	00000154	25 APR 73 16:34:54	CMP2.3A0
FAS$PAS	000030	00000155	24 JUN 73 23:04:30	CMP2.4A0
FAS$PAT	000033	00000157	24 JUN 73 23:04:31	CMP2.4A0
FAS$POS	000013	00000161	25 APR 73 16:35:00	CMP2.3A0
FAS$PTP	000032	00000162	24 JUN 73 23:04:32	CMP2.4A0
FAS$RAR	000022	00000164	31 MAY 73 19:54:02	CMP2.4A0
FAS$REM	000003	00000165	24 JUN 73 23:04:35	CMP2.4A0
FAS$RET	000055	00000166	24 JUN 73 23:04:35	CMP2.4A0
FAS$RSG	000004	00000170	25 APR 73 16:35:11	CMP2.3A0
FAS$RTB	000005	00000171	24 JUN 73 23:04:40	CMP2.4A0
FAS$RTS	000034	00000172	25 APR 73 16:35:12	CMP2.3A0
FAS$SIZ	000006	00000173	25 APR 73 16:35:13	CMP2.3A0
FAS$SKW	000013	00000174	24 JUN 73 23:04:41	CMP2.4A0
FAS$SNP	000062	00000175	24 JUN 73 23:04:43	CMP2.4A0
FAS$SPN	000024	00000177	24 JUN 73 23:04:44	CMP2.4A0
FAS$SRS	000025	00000200	24 JUN 73 23:04:44	CMP2.4A0
FAS$SSG	000021	00000201	25 APR 73 16:35:20	CMP2.3A0
FAS$SSR	000037	00000202	04 AUG 73 16:16:03	CMP2.4A0
FAS$STE	000126	00000204	04 AUG 73 16:16:05	CMP2.4A0
FAS$STI	000034	00000207	25 APR 73 16:35:25	CMP2.3A0
FAS$STR	000065	00000210	25 APR 73 16:35:46	CMP2.3A0
FAS$SYM	000153	00000212	24 JUN 73 23:04:48	CMP2.4A0
FAS$TAB	000010	00000215	24 JUN 73 23:04:52	CMP2.4A0
FAS$TRM	000052	00000216	25 APR 73 16:35:54	CMP2.3A0
FAS$VAL	000005	00000220	25 APR 73 16:35:56	CMP2.3A0
FAS$VPR	000031	00000221	31 MAY 73 19:54:20	CMP2.4A0
PRINTRG	000131	00000222	25 APR 73 16:35:58	CMP2.3A0

99 DECKS IN LIBRARY

APPENDIX III

Sample Programs

```
             *
             *                      TOPOLOGICAL SORT
             *             MAPS A PARTIAL ORDERING OF OBJECTS INTO A LINEAR ORDERING
             *
 1                         STTIME     = TIME()
 2                         PAIR       = BREAK(↑<↑) . MU LEN(1) BREAK(↑,↑) . NU LEN(1)
 3                         DATA(↑ITEM(COUNT,TOP)↑)
 4                         DATA(↑NODE(SUC,NEXT)↑)
 5                         DEFINE(↑INDEX(TAU)↑)
             *
             *                      READ IN THE NUMBER OF ITEMS, N, AND GENERATE AN
             *                      ARRAY OF ITEMS.
             *             EACH ITEM HAS TWO FIELDS, (COUNT,TOP), WHERE
             *                      COUNT = NO. OF ELEMENTS PRECEDING IT.
             *                      TOP = TOP OF LIST OF ITEMS SUCCEEDING IT.
             *
 6                         N          = TRIM(INPUT)
 7                         X          = ARRAY(↑0/↑ N)
             *
             *                      INITIALIZE THE ITEMS TO (0,NULL).
             *
 8 T1                      X[I]       = ITEM(0,)                    :F(T1A)
 9                         I          = I + 1                       :(T1)
             *
             *                      READ IN RELATIONS
             *
10 T1A                     OUTPUT     = ↑ THE RELATIONS ARE ↑
11 T2A                     REL        = TRIM(INPUT) ↑,↑             :F(T3A)
12                         OUTPUT     = ↑ ↑ REL
13 T2                      REL        PAIR =                        :F(T2A)
14                         J          = INDEX(MU)
15                         K          = INDEX(NU)
             *
             *                      SINCE MU < NU, INCREASE THE COUNT OF THE KTH ITEM
             *                      AND ADD A NODE TO THE LIST OF SUCCESSORS OF THE
             *                      JTH ITEM
             *
16 T3                      COUNT(X[K]) = COUNT(X[K]) + 1
17                         TOP(X[J])  = NODE(K,TOP(X[J]))           :(T2)
             *
             *                      INITIALIZE THE QUEUE FOR OUTPUT
18 T3A                     R          = 0
19                         COUNT(X[0]) = 0
20                         K          = 0
21 T4                      K          = LT(K,N) K + 1               :F(T4A)
22                         COUNT(X[R]) = EQ(COUNT(X[K]),0) K        :F(T4)
23                         R          = K                           :(T4)
24 T4A                     F          = COUNT(X[0])
             *
             *             OUTPUT THE FRONT OF THE QUEUE
             *
25                         OUTPUT     = ↑ THE LINEAR ORDERING IS ↑
26 T5                      OUTPUT     = NE(F,0) ↑ ↑ $(F ↑/↑)        :F(T8)
```

```
27           N             = N - 1
28           P             = TOP(X[F])
      *
      *                    ERASE RELATIONS
      *
29  T6       IDENT(P)                                          :S(T7)
30           Y             = COUNT(X[SUC(P)])
31           COUNT(X[SUC(P)]) = GT(Y,1) Y - 1                  :S(T6A)
32           COUNT(X[SUC(P)]) = 0
      *
      *                    IF COUNT IS ZERO ADD ITEM TO QUEUE.
      *
33           COUNT(X[R]) = SUC(P)
34           R             = SUC(P)
35  T6A      P             = NEXT(P)                           :(T6)
      *
      *                    REMOVE FROM QUEUE.
      *
36  T7       F             = COUNT(X[F])                       :(T5)
      *
      *                    FUNCTION DEFINITIONS.
37  INDEX    INDEX         = DIFFER($(TAU ↑/↑)) $(TAU ↑/↑)     :S(RETURN)
38           TERMCT        = LT(TERMCT,N) TERMCT + 1          :F(FRETURN)
39           INDEX         = TERMCT
40           $(TERMCT ↑/↑) = TAU
41           $(TAU ↑/↑) = TERMCT                               :(RETURN)
      *
42  T8       OUTPUT        = NE(N,0) ↑ THE ORDERING CONTAINS A LOOP↑
43           OUTPUT        = ↑   EXECUTION TIME WAS ↑ TIME() - STTIME ↑ MS.↑


19 VARIABLES ENCOUNTERED
46 CONSTANTS SEEN
8 FIXS MADE


*TOTAL COMPILATION TIME:5637 MS., 0 ERROR DIAGNOSTICS*
```

```
THE RELATIONS ARE
LETTERS<ALPHANUM,NUMBERS<ALPHANUM,
BLANKS<OPTBLANKS,
NUMBERS<REAL,
NUMBERS<INTEGER,
LETTERS<VARIABLE,ALPHANUM<VARIABLE,
BINARY<BINARYOP,BLANKS<BINARYOP,
UNQALPHABET<DLITERAL,
UNQALPHABET<SLITERAL,
SLITERAL<LITERAL,DLITERAL<LITERAL,INTEGER<LITERAL,REAL<LITERAL,
THE LINEAR ORDERING IS
LETTERS
NUMBERS
BLANKS
BINARY
UNQALPHABET
INTEGER
REAL
ALPHANUM
OPTBLANKS
BINARYOP
SLITERAL
DLITERAL
VARIABLE
LITERAL
  EXECUTION TIME WAS 151 MS.
```

FASBOL COMPILER VER 0.2 08/30/73 13:13:21

```
                 ****************************************************************************
                 *                                                                          *
                 *           THIS PROGRAM COMPUTES AND PRINTS A TABLE OF N FACTORIAL        *
                 *           FOR VALUES OF N FROM 1 THROUGH AN UPPER LIMIT 'NX'.            *
                 *                                                                          *
                 *           IT DEMONSTRATES A METHOD OF MANIPULATING NUMBERS WHICH ARE     *
                 *           TOO LARGE FOR THE COMPUTER, AS STRINGS OF CHARACTERS. THE      *
                 *           COMMAS IN THE PRINTED VALUES ARE OPTIONAL, ADDED FOR READING   *
                 *           EASE.                                                          *
                 *                                                                          *
                 ****************************************************************************
                 *
                 *           INITIALIZATION
                 *
       1                     NX          = 35
                 *
       2                     N           = 1
       3                     NSET        = 1
       4                     NUM         = ARRAY(1000)
       5                     NUM[1]      = 1
       6                     FILL        = ARRAY('0/3')
       7                     FILL[0]     = '000'
       8                     FILL[1]     = '00'
       9                     FILL[2]     = '0'
                 *
      10                     OUTPUT      = '           TABLE OF FACTORIALS FOR 1 THROUGH ' NX
      11                     OUTPUT      =
                 *
                 *           COMPUTE THE NEXT VALUE FROM THE PREVIOUS ONE.
                 *
      12 L1                  I           = 1
      13 L2                  NUM[I]      = NUM[I] * N                       :F(ERR)
      14                     I           = LT(I,NSET) I + 1                 :S(L2)
      15                     I           = 1
      16 L3                  LT(NUM[I],1000)                                :S(L4)
      17                     NUMX        = NUM[I] / 1000                    :F(ERR)
      18                     NUM[I + 1] = NUM[I + 1] + NUMX                 :F(ERR)
      19                     NUM[I]      = NUM[I] - 1000 * NUMX             :F(ERR)
      20 L4                  I           = LT(I,NSET) I + 1                 :S(L3)
                 *
                 *           FORM A STRING REPRESENTING THE FACTORIAL.
                 *
      21 L5                  NSET        = DIFFER(NUM[NSET + 1]) NSET + 1
      22                     NUMBER      = NUM[NSET]                        :F(ERR)
      23                     I           = GT(NSET,1) NSET - 1              :F(L7)
      24 L6                  NUMBER      = NUMBER ',' FILL[SIZE(NUM[I])] NUM[I]
      25                     I           = GT(I,1) I - 1                    :S(L6)
                 *
                 *           OUTPUT A LINE OF THE TABLE
                 *
      26 L7                  OUTPUT      = N '!=' NUMBER
      27                     N           = LT(N,NX) N + 1                   :S(L1)F(END)
                 *
```

```
000054             *          ERROR TERMINATION
000055             *
000056          28 ERR        OUTPUT = N ≠↑ CANNOT BE COMPUTED BECAUSE OF TABLE OVERFLOW.≠
000057          29            OUTPUT = ↑     INCREASE THE SIZE OF ARRAY  NUM .↑
000058             *
000059          30 END

10 VARIABLES ENCOUNTERED
33 CONSTANTS SEEN
5 FIXS MADE


 *TOTAL COMPILATION TIME:3500 MS., 0 ERROR DIAGNOSTICS*
```

TABLE OF FACTORIALS FOR 1 THROUGH 35

1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
7!=5,040
8!=40,320
9!=362,880
10!=3,628,800
11!=39,916,800
12!=479,001,600
13!=6,227,020,800
14!=87,178,291,200
15!=1,307,674,368,000
16!=20,922,789,888,000
17!=355,687,428,096,000
18!=6,402,373,705,728,000
19!=121,645,100,408,832,000
20!=2,432,902,008,176,640,000
21!=51,090,942,171,709,440,000
22!=1,124,000,727,777,607,680,000
23!=25,852,016,738,884,976,640,000
24!=620,448,401,733,239,439,360,000
25!=15,511,210,043,330,985,984,000,000
26!=403,291,461,126,605,635,584,000,000
27!=10,888,869,450,418,352,160,768,000,000
28!=304,888,344,611,713,860,501,504,000,000
29!=8,841,761,993,739,701,954,543,616,000,000
30!=265,252,859,812,191,058,636,308,480,000,000
31!=8,222,838,654,177,922,817,725,562,880,000,000
32!=263,130,836,933,693,530,167,218,012,160,000,000
33!=8,683,317,618,811,886,495,518,194,401,280,000,000
34!=295,232,799,039,604,140,847,618,609,643,520,000,000
35!=10,333,147,966,386,144,929,666,651,337,523,200,000,000