SPL/M

Reference Manual

PROGRAMMA

Software
Program
Products

6800.002

PROGRAMMA
INTERNATIONAL, Inc.
3400 Wilshire Boulevard
Los Angeles, CA 90010

(213) 384-0579

PROGRAMMA

S P L / M C O M P I L E R

REFERENCE MANUAL

VER 1.2

JUNE 1979

JUNE 1979 EDITION

This edition (6800.002) is a major revision and obsoletes all previous editions and documents.

Technical changes are marked with a bar in the outer margin. Changes due to subsequent releases will be documented in the future publication bulletins or revisions.

Requests for copies of PROGRAMMA publications should be made to your PROGRAMMA representative or to the PROGRAMMA central office.

A reader's comment form is provided at the back of this publication. If the form has been removed, comments may be addressed to:

PROGRAMMA International, Inc.
Publications Department
P.O. Box 70279
Los Angeles, CA 90070

TABLE OF CONTENTS

XII. APPENDICES

I. INTRODUCTION

SPL/M (Small Programming Language for Microprocessors) is based on the language PL/M, initially developed by the Intel Corporation.

SPL/M is a block-structured language which features arbitrary length identifiers and structured programming constructs. It is suitable for systems programming on small computers, since the compiler requires only 20K of memory to run. Either two cassette decks or a disk are also required.

The language can be compiled in only one pass, which means that the source code has to be read only once.

Unlike most high-level language translators available for microprocessors, SPL/M is a true compiler: it generates absolute 6800 object code which requires no run-time interpreter. Due to extensive intra-statement optimization, the generated code is almost as efficient as the equivalent assembly language.

The compiler has a number of compile-time options, including a printout that contains the interlisted object code. Syntactical error messages use position indicators to indicate exactly where an error occurs.

This manual has been organized to be usable as both a tutorial and a reference guide. In addition to the many examples in the text, a complete SPL/M program is presented in Appendix C.

As an example of the type of application SPL/M is suited for, this entire manual was formatted using a text processing system written in 800 lines of SPL/M.

Some details of the compiler implementation are presented in the paper "SPL/M - A Cassette-Based Compiler", by Thomas W. Crosley, in the Conference Proceedings, Second West Coast Computer Faire, March, 1978.

II. PRIMITIVES

An SPL/M program consists of primitives (reserved words, identifiers, and constants), along with special characters (operators).

One or more blanks (spaces) are required between any two primitives on the same line, to tell them apart. Blanks are allowed anywhere else, except in the middle of a primitive or a two character operator (such as >=). A carriage return is treated the same as a blank; therefore statements can spill over onto as many lines as necessary.

Comments may be embedded in an SPL/M program anywhere a blank is legal. Comments are delimited by a /* ... */ pair:

```
/* COMMENTS MAY GO OVER
   MORE THAN ONE LINE */
```

Identifiers

An identifier is a programmer assigned name for a variable, procedure, or symbolic constant. Identifier names may be up to 31 characters long.

The first character must be alphabetic (A-Z), while the remaining characters may be either alphanumeric (A-Z, 0-9) or the separation character ($). The latter is completely ignored by the compiler: an identifier with imbedded $'s is equivalent to the same identifier with the $'s omitted.

Examples of valid identifiers:

```
ACIANO      ACIA$NO      (same variable)
BUFFER1
A$RATHER$LONG$PROCEDURE$NAME
```

Identifier names must not conflict with the reserved words of SPL/M, such as DECLARE, PROCEDURE, etc. A complete list of reserved words for both Versions 1 and 2 of SPL/M is provided in Appendix D.

All identifers must be declared before they are referenced. Variables and symbolic constants are defined via the DECLARE statement (Section V); procedures are defined via the PROCEDURE statement (Section VII).

III. DATA REPRESENTATIONS

Constants

Constants can be either a number or a character string. As their name implies, their value remains constant during program execution.

A numeric constant, or number, is a string of digits representing an unsigned integer in the range 0-65535. A number is assumed to be decimal unless it is terminated by the letter H, indicating hexadecimal. The first character of a hexadecimal constant must always be numeric (a leading zero is always sufficient).

Examples of numeric constants:

0	32	65535
10	20H	0FFFFH
0AH		

A character constant, or string, consists of one or more ASCII characters enclosed in apostrophes. A null string (i.e. '') is not permitted. Imbedded apostrophes are represented by two consecutive apostrophes (e.g. DON''T).

Constants of one or two characters are equivalent to the numeric constant representing the ASCII code for the character(s). In a two character constant, the left-most character is placed in the most significant byte.

Character constants of more than two characters may only appear in a DATA declaration (Section V).

Examples of character constants:

'A' = 41H
' ' = 20H
'12' = 3132H
'''' = 27H (one ')

'THIS IS A LONG STRING'

Variables

Variables are memory locations set aside by the programmer to hold data that changes during the execution of a program. Variables can be declared as either type BYTE (8 bit data) or type ADDRESS (16 bit data). BYTE variables should be used wherever possible to avoid the overhead associated with double precision arithmetic on the 6800.

Variables are defined using the DECLARE statement (Section V), e.g.

```
DECLARE CTR BYTE;
DECLARE BUFOPTR ADDRESS;
```

Vectors (one dimensional arrays) can also be declared, e.g.

```
DECLARE LIST (10) BYTE;
```

which sets aside 10 bytes of storage. A vector has n elements, referenced as

V(0), V(1), ..., V(n-1)

The value in parentheses is the subscript, which can be any SPL/M expression (Section IV). The subscript is added to the base address for BYTE vectors to generate the correct memory reference. For ADDRESS variables, twice the subscript is added to the base to generate the correct memory reference.

For example, if the BYTE vector LIST declared above was located at memory address 400, then LIST(4) would refer to memory address 404. However if LIST was an ADDRESS vector, then LIST(4) would refer to memory addresses 408 and 409.

Subscripted variables can be used anywhere a variable is allowed in SPL/M, except as the operand of the dot operator (Section IV).

The first element of a vector may also be referenced without the subscript; i.e. V and V(0) are the same.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

IV. EXPRESSIONS AND ASSIGNMENT STATEMENTS

An expression is simply a way of computing a value. Expressions are formed by combining operators (such as + or *) with either operands (variables or constants) or other expressions enclosed in parentheses.

An arithmetic expression consists of one or more operands which are combined using the following arithmetic operators:

+	addition
-	subtraction (unary minus also allowed)
*	unsigned multiplication
/	unsigned integer division
MOD	modulo (remainder from a division)
.	dot operator (see below)

Examples:

```
X
ALPHA - BETA
10 MOD 3        (result =1)
-1
X*(Y+Z)/2
.BUF1
```

The unary dot operator (.) generates a numeric constant equal to the memory address of a variable. The variable cannot have a subscript.

A relational expression consists of two arithmetic expressions combined with one of the following relational operators:

<	less than
<=	less than or equal to
=	equal to
<>	not equal to
>=	greater than or equal to
>	greater than

Comparisons are always performed assuming the operands are unsigned integers. If the specified relation holds, a value of OFFH (true) is returned; otherwise the result is 0 (false).

Examples:

```
A > 1
CNTR <= LIMIT+OVER
LOOP<>0
```

A logical expression consists of either arithmetic or relational expressions combined with one or more of the following logical operators:

OR	bitwise OR
XOR	bitwise exclusive OR
AND	bitwise AND
NOT	1's complement (unaryoperator)

Examples:

```
LADIES AND GENTLEMEN
NOT FLAGS          (same as FLAGS XOR -1)
X > 1 OR Y < 2
```

The following table summarizes the effect of each logical operator:

X	Y	X OR Y	X XOR Y	X AND Y	NOT X
0	0	0	0	0	1
0	1	1	1	0	1
1	0	1	1	0	0
1	1	1	0	1	0

Logical expressions are used in assignment statements to perform bit manipulation, and in IF and DO-WHILE statements (Section VI) to specify a series of conditional tests.

Operator Precedence

The order of evaluation of operators in an expression is primarily determined by operator precedence.

Operands are associated with the adjacent operator of highest precedence. Operands adjacent to two operators of equal precedence may be associated with either one. Operators with the highest precedence are evaluated first. Two operators of the same precedence may be evaluated in either order.

The following list summarizes the operator precedence for SPL/M:

```
highest:  ( )  .
          unary -
          *  /  MOD
          +  -
          =  <  >  <>  <=  >=
          NOT
          AND
lowest:   OR  XOR
```

Since parentheses have the highest precedence, they can be used to override the implicit order of evaluation. The following fully parenthesized expression

```
IF (A=3) OR (B > (10*(I+1))) THEN
```

can also be written:

```
IF A=3 OR B>10*(I+1) THEN
```

The parentheses around the I+1, to force the addition to be done first, are the only ones required in this case.

Assignment Statements

Assignment statements perform the real work of a program. They are used to assign the result of an expression to a variable location. The format is:

```
variable = expression;
```

The value of the variable on the left-hand side of the equal sign is replaced by the value of the expression on the right-hand side.

Examples:

```
CTR = CTR + 1;
LIST(I) = 0;
```

Implicit Type Conversions

Mixed mode is a situation which arises when BYTE and ADDRESS variables or constant are combined in the same expression or assignment statement. To avoid generating unexpected results, SPL/M attempts to use double-precision arithmetic throughout mixed mode expressions.

As soon as an ADDRESS variable or constant is encountered (scanning from left to right), then the remainder of the statement or expression is evaluated in double-precision mode. For example, if X is an ADDRESS variable, then

```
X = -1;
```

will set X = OFFFFH since the unary subtraction will be carried out in double precision.

When operating in double-precision mode, the high-order eight bits of any BYTE variables or constants in an expression are assumed to be 0. In an assignment statement, if the variable on the left-hand side is type BYTE, whereas the expression on the right-hand side is type ADDRESS, then the high-order eight bits of the expression will be lost.

In a complex relational expression involving ADDRESS variables on one side and BYTE variables on the other, the ADDRESS variables should appear first to force the entire expression to be evaluated in double-precision.

Note: the rules used by SPL/M for evaluating mixed-mode expressions are not the same as PL/M.

Functions for performing explicit type conversions are also available in SPL/M; see Section VIII.

V. DECLARATIONS

Variables, constant data arrays, and symbolic constants are defined using the DECLARE statement. (DCL is an allowed abbreviation for DECLARE). All programmer-defined identifiers must be declared before they are referenced in the program. Declarations are subject to "scope", which is explained under program organization (Section IX).

Variable Declarations

The general form of the declare statement is:

```
DECLARE identifier [(bounds)] type;
```

where "(bounds)" is optional and is used only for vector declarations (see below). The "type" may be either BYTE, denoting 8-bit data, or ADDRESS (abbreviated ADDR), denoting 16-bit data.

Examples:

```
DECLARE CTR BYTE;
DCL BUF$PTR ADDRESS;
```

Vectors (one-dimensional arrays) are defined by specifying the number of elements following the variable name; e.g.

```
DCL LIST (10) BYTE;
```

which sets aside 10 bytes of storage, and

```
DCL A$LIST (10) ADDR;
```

which allocates 20 bytes (two for each address element). Vectors are referenced using subscripts as explained in Section III.

The number of elements in a vector declaration may be zero, in which case no storage is reserved. The variable will refer to the same memory location as the next data declaration. For example,

```
DCL BIG$CTR (0) ADDR,
    HIGH$CTR BYTE,
    LOW$CTR BYTE;
```

HIGH$CTR and LOW$CTR overlay the high and low bytes of BIG$CTR. This example also shows how several variables can be declared in the same statement. Each declaration is separated by a comma.

Sometimes it is desirable to declare a variable at a particular memory location. This is done by preceding the DECLARE statement with an origin, which will cause the next BYTE or ADDRESS variable to be allocated at the given address. Origins consist of a number followed by ':'. For example,

```
38H: DCL ACIA$NO ADDR, NO$PRNT BYTE;
3CH: DCL BUF$BEG ADDR;
     DCL BUF$END ADDR;
```

will cause the following allocations to take place:

```
38H-39H     ACIANO
3AH         NOPRNT
3CH-3DH     BUFBEG
3FH-3FH     BUFEND
```

If a declaration is not preceded by an origin, variables are allocated storage immediately following the last declaration. Unless overridden by an explicit origin, the first variable declaration starts at 10H. Declare origins have no effect on DCL DATA and DCL LIT statements (discussed below); however an origin on either will affect the next variable allocation.

Constant Data Declarations

It is often necessary to define constant data, such as character strings or a table. This is done via a DECLARE DATA statement, which has the general form:

```
DECLARE identifier DATA (constant list) ;
```

where "constant list" is a list of numeric or character constants, separated by commas.

It is assumed that data declared in this way will not change during execution of the program. The data is located within the program object code.

The identifier defined in a DCL DATA statement is always of type byte, and is referenced using subscripts the same as any vector.

Examples:

```
DECLARE REVERSE$DIGITS DATA (9,8,7,6,5,4,3,2,1,0);

DCL MSG DATA ('A MESSAGE STRING',4);
```

Symbolic Constant Declaration

The DECLARE LITERALLY statement provides a compile-time symbolic constant substitution mechanism similar to the "equate" facility in assemblers. The general form is:

```
DECLARE identifier LITERALLY 'number';
```

LITERALLY may be abbreviated as LIT. Whenever the identifier is encountered in the program, it will be replaced by the number.

Examples:

```
DECLARE CASS1 LITERALLY '0F050H';
DCL TRUE LIT '0FFH', FALSE LIT '0';
                .
                .
                .

IF DECK <> CASS1 THEN
   DEFAULT = FALSE;
```

VI. FLOW OF CONTROL & GROUPING

Various SPL/M statement types are used to alter the path of program execution. SPL/M does not have the GOTO statement available in BASIC and FORTRAN. However the structured programming constructs (IF-THEN-ELSE, DO-END, and DO-WHILE) can be used to express any program more clearly than if GOTO's were used.

IF Statement

The IF statement selects alternate execution paths, based on a conditional test. IF statements have two forms:

a) IF expression THEN statement-1;

b)
```
IF expression
     THEN statement-1;
     ELSE statement-2;
```

Execution of an IF statement begins by evaluating the expression following the IF. If the right-most (least significant) bit of the result is a 1, then statement-1 is executed. If the bit is a 0, no action is taken for the first form (a), and statement-2 is executed for the second form (b).

Since the result of a relational expression is either OFFH (true) or 0 (false), the construction "IF relational-expr THEN" has the expected result.

In the second form of the IF statement above (b), statement-1 may not be an IF statement. This avoids any ambiguity in the following construction:

```
IF expression
   THEN IF expression
        THEN statement-1;
        ELSE statement-2;
```

The rule in this case is that the ELSE belongs to the second (innermost) IF statement. If needed, a DO-END group (defined below) can be used to associate the ELSE with the first IF statement:

```
IF expression
   THEN DO;
        IF expression THEN statement-1;
   END;
   ELSE statement-2;
```

The ELSE now clearly belongs to the first IF. The following are examples of IF statements:

```
IF CFLAG THEN CTR = CTR+1;

IF A > 0 AND B > 0
   THEN A=B;

IF X>0 THEN Y=1; ELSE Y=2;
```

DO-END Groups

The DO-END statement is used to group together a sequence of SPL/M statements, such that they are treated as a single executable statement in the flow of control. For example,

```
IF SWITCH
   THEN DO;
        TEMP=A;
        A=B;
        B=TEMP;
   END;
```

All three statements in the DO-END group will be executed if the variable SWITCH is true. Note that indentation is usually used with IF and DO statements to make the logic of the program stand out.

Simple DO-END groups are also used (less frequently) to create a block in which local variables are declared, as described in Section IX.

DO-WHILE Statement

The DO-WHILE statement causes a group of statements to be repeatedly executed as long as a condition is satisfied. The general form is:

```
DO WHILE expression;
   statement-1;
        .
        .
        .
   statement-n;
END;
```

The statements within the DO-WHILE are executed as long as the result of the expression has its right-most bit equal to 1. The expression is evaluated at the beginning of each execution cycle.

This version of SPL/M does not have the PL/M iterative-type DO (like the FOR statement in BASIC). However the more general DO-WHILE can be used in an identical manner:

```
I = 0;
DO WHILE I < 10;
   CHAR = I+'0';
   CALL PUTCHR;  /* DISPLAY 0-9 */
   I = I+1;
END;
```

It is sometimes desirable to terminate the execution of a DO-WHILE abnormally (i.e. for some condition other than the expression following the DO). This is facilitated by the BREAK statement, which causes a transfer of control to the first statement following the END which terminates the innermost DO-WHILE.

Example:

```
I = 0; FOUND = 0;
DO WHILE NOT FOUND;
   IF LIST(I) = KEY              /* SEARCH LIST FOR KEY */
      THEN FOUND = 1;            /* EXIT NEXT CYCLE */
   ELSE DO;
      I = I+1;
      IF I >= 100 THEN BREAK; /* ABNORMAL EXIT */
   END;
END;
```

If the key is found in the list, the DO-WHILE will exit normally with FOUND=1 and I equal to the list index. Otherwise the BREAK will terminate abnormally with FOUND=0.

Note: the BREAK statement is an SPL/M extension and is not in PL/M.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

VII. PROCEDURES

Well designed programs make frequent use of subroutines, each of which is related to a particular function. In SPL/M, subroutines are called procedures, and are defined as follows:

```
label: PROCEDURE;
       statement-1;
            .
            .
            .
       statement-n;
END;
```

The "label" is the procedure name, which is required later when the procedure is called. PROCEDURE may be abbreviated PROC.

In this version of SPL/M, all procedures must be defined at the beginning of the program (see Section IX) and nesting of procedure definitions is not allowed.

Since a procedure is a block (also discussed in Section IX), all variables declared within it are "local" and cannot be referenced outside of the procedure. All storage declared in SPL/M is static. Automatic stacking of local variables is not done on entry to a procedure.

All values passed to and from procedures must be done via global variables since procedures cannot have parameters in this version of SPL/M.

CALL Statement

Procedures are invoked by the CALL statement:

```
CALL procedure-name;
```

where the procedure must have been previously defined as described above.

Example:

```
DCL MAX$LINE LITERALLY '80';
DCL LINE (MAX$LINE) BYTE;  /* GLOBAL */
        .
        .
        .

CLEAR$LINE: PROCEDURE;
   DCL I BYTE;  /* LOCAL */
   I=0;
   DO WHILE I < MAX$LINE;
      LINE(I) = ' ';
      I = I+1;
   END;
END;
        .
        .
        .

CALL CLEAR$LINE;
```

It is also possible to call a procedure by its address. This makes it easier to link to assembly language subroutines in an operating system. For example,

```
CALL OFC37H;   /* HOME CURSOR */
CALL OFC3DH;   /* CLEAR SCREEN */
```

Note: the construction "CALL number" is an SPL/M extension and is not in PL/M.

The "declare literally" facility (Section V) can be used to define the address as a symbolic constant to keep the reference symbolic:

```
DCL HOME LIT 'OFC37H';
        .
        .
        .
CALL HOME;
```

RETURN Statement

When a procedure is called, it starts execution at the beginning of the procedure and normally does not return until the END matching the PROCEDURF statement is reached. However it is possible to force an earlier return by using the RETURN statement, e.g.

```
IF ERROR THEN RETURN;
```

Whether a RETURN statement is used or not, a procedure returns to the statement following the original CALL.

VIII. MISCELLANEOUS FACILITIES

Direct References to Memory

It is sometimes desirable to refer to the memory address space of the 6800 directly. (In fact this is the only way I/O can be performed directly in SPL/M, since the language does not have explicit input/output statements. But I/O is usually done via calls on existing operating systems routines.)

When required, direct reference to memory can be done using the MEM and MEMA vectors, which are predeclared to start at address 0. MEM is type byte, while MEMA is type address. The normal doubling of subscripts is not done for MEMA; for example

```
MEMA(38H) = 0F050H;
```

sets memory locations 38H and 39H to the hexadecimal value 0F050H.

Note: MEM and MEMA are SPL/M extensions and are not in PL/M.

When used on the left-hand side of an assignment statement, MEM is like the POKE function in some BASIC's. On the right-hand siide, MEM is like the PEEK function.

The subscript can be any arithmetic expression, but usually is just an address variable. In the following byte move subroutine, global variables BUF1 and BUF2 contain the start addresses of two buffers, and BSIZE is the number of bytes to move:

```
BYTE$MOVE: PROC;
   DO WHILE BSIZE <> 0;
      MEM(BUF2) = MEM(BUF1);
      BUF1 = BUF1+1; BUF2 = BUF2+1;
      BSIZE = BSIZE-1;
   END;
END;
```

Explicit Type Conversion

Section V discussed implicit (automatic) type conversions in mixed mode expressions. SPL/M also provides two explicit type conversions in the form of built-in functions, which take address expressions as arguments. The functions may appear anywhere an expression is legal.

LOW(expr) returns the least-significant byte of its argument.

HIGH(expr) returns the most-significant byte of its argument.

GENERATE Statement

It is occasionally necessary to link to operating system subroutines which pass values in registers. The GENERATE statement can be used to produce machine code "patches" to accomplish this. It generates code in-line wherever it appears in an SPL/M program. Because of the low-level nature of this statement, and the possibility of making errors, it should be used only where absolutely necessary.

The GENERATE statement has the form:

```
GENERATE (constant list);
```

where "constant list" is a list of numeric, character, or symbolic constants, including address (dot) references. GENERATE may be abbreviated GEN.

Note: the GENERATE statement is an SPL/M extension and is not in PL/M.

The following example stores the contents of the accumulator at location 42H after calling a subroutine to input a character:

```
CALL OFC4AH;
GEN(97H, 42H);
```

However using only hexadecimal constants makes the code nearly impossible to read. This can be improved by using DCL LIT's and declaring a variable at address 42H:

```
42H: DCL CHAR BYTE;
DCL GET$CHAR LIT '0FC4AH',
    STAA LIT '97H';
          .
          .
          .

CALL GET$CHAR;
GEN (STAA, .CHAR);
```

For additional examples, refer to the SPL/M library routines presented in Appendix B.

IX. PROGRAM ORGANIZATION AND SCOPE

In general, an SPL/M program consists of a set of global declarations, followed by any procedure declarations, followed by the "main" portion of the program. The last line of the program must contain the characters EOF (end of file) which generates an RTS instruction to return to the caller of the main program.

DECLARE statements may appear anywhere in SPL/M, but their location may have different effects due to the "scoping" rules discussed below. In all cases, all names, whether they are variables, procedures, or symbolic constants, must be defined before they are referenced in the program.

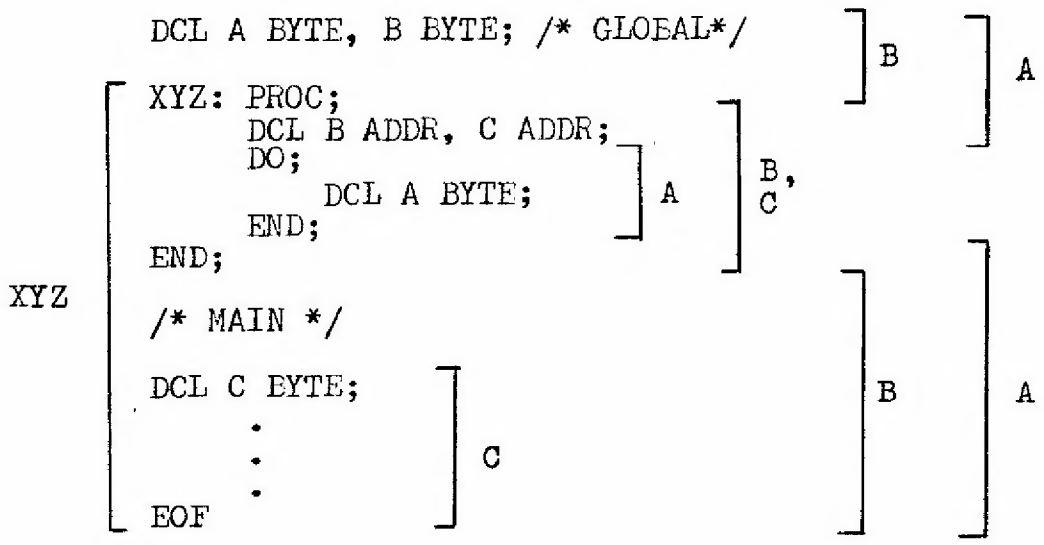
Block Structure and Scope

The largest syntactic unit in an SPL/M program is the outermost program block, which consists of the global declarations, procedure definitions, and the "main" program.

Global declarations will be known, or available, to all procedures and the main program. Each procedure may also contain its own declarations, which are local; i.e. known only within that procedure.

Procedures and/or the main program may also have DO-END groups (Section VI) containing additonal declarations, which are local to each group.

Example:



The brackets indicate the "scope" of each variable.

Variables, once defined, can be redefined only within a nested block (procedure or DO-END group), which will result in additional static storage being allocated. The new definition is known only within the nested block(s); when the end of the nested block is reached the original definition is in effect again.

Variables, unless redefined, are known within the block in which they are declared and in all blocks nested within it.

Program Origins

Origins, which are simply a number followed by ':', have already been discussed in the context of declare statements (Section V).

A program origin is any origin not preceding a DECLARE statement. Program origins affect the generation of the next byte of object code, including DCL DATA constants (which are located within the program object module).

In this version of SPL/M, program origins are restricted to the following locations:

1) First statement of a program (defines starting address).

2) Beginning of each procedure definition (the origin must be placed just ahead of the procedure name).

3) First statement of "main" (allowed only if the program contains procedure definitions).

In all the cases above, origins are optional. In the absence of any origin the first byte object code will start at location 100H. If the main program or a procedure lacks an origin, the associated code will follow the code immediately preceding.

If provided, the initial (start) origin must be immediately followed by a "null statement" (e.g. 0A100H:;) to distinguish it from a declare origin.

When an origin is specified, the user is responsible for insuring that the resulting code does not overlap code that has already been generated.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

The following example summarizes the SPL/M program organization. Everything in brackets [] is optional; and any addresses are for example only. Note that declares can go anywhere; however for clarity it is best to restrict them to the beginning of the program, the beginning of each procedure, and the beginning of "main".

```
 [ 200H:; ]                          /* OPT. START ADDRESS */
[[ 42H: ] DCL's ]                    /* GLOBAL DECLARES */
[[ 300H: ] XYZ: PROC;                /* OPT. PROCEDURE
                 .                      DEFINITIONS */
                 .
                 .
  END;                               ]
 [ 400H: ]                           /* OPT. ORIGIN FOR MAIN */
      .
      .
  /* main */
      .
      .
EOF
```

A jump from the beginning of the program (e.g. 200H) to the beginning of the code for main (e.g. 400H) is automatically generated if there are procedure definitions and if there is either an explicit start address provided or there are any global DCL DATA's.

Refer also to Appendix C for an example of a complete SPL/M program that contains many of the elements described above.

X. COMPILE AND CONFIGURATION OPTIONS

(FLEX Version 1.2)

System Considerations

This version of the compiler is designed to run on a 6800-based system, such as the SWTPc, running under the FLEX Operating System. In particular, it assumes the existence of:

FLEX 1.0 or 2.0 (not miniFLEX)
20K of user RAM starting at location 0000
SWTBUG monitor ROM or equivalent

Compiler Disk

The disk supplied with the compiler contains the following files:

SPLM.CMD	- SPL/M compiler
FLX102.TXT	- Assembler source for compiler interfaces
SPLM.LIB	- SPL/M library (general DOS interfaces)
SPLMRFAD.LIB	- SPL/M library (reading sequential files)
SPLMWRIT.LIB	- SPL/M library (writing sequential files)
SIZE.TXT	- SPL/M source for sample program (SIZE)

The SIZE.TXT source file is intended to be used as a test of the compiler. It also brings in two of the library files using the #INCLUDE facility discussed below.

Running the Compiler

The compiler has several compile-time options which control the generation of listings and binary files.

The general syntax for the SPLM command is:

SPLM[,<source>[,<binary>][,+<option list>]]

The '<>' enclose a field defined below and are not actually typed. The '[]' surround optional fields.

All parameters are optional. If none are provided, then the compiler runs interactively with the source input coming directly from the keyboard. This is useful for experimenting, to see what kind of code the compiler generates for a particular input. In

this mode a full code listing is always output to the terminal. A binary object file is not produced.

The normal mode however is for a <source> file name to be specified to be compiled. In this case the compiler reads the named file from disk until an EOF statement is encountered in the source. The defaults for the <source> file specification are a .TXT extension and the working drive number.

If the optional <binary> file name is also specified, it is used as the name of the object file written to disk. If <binary> is not included in the command, the binary file will have the same 'name' as the source file, but with a .BIN extension.

The option list is prefixed with a plus sign ('+'), with each option represented by a single letter. The letters may be in any order. The following options are available:

B (No binary). Do not create a binary file on disk, even if a <binary> file name is specified.

Y (Yes, delete). Delete an old binary file of the same name as the one about to be produced. If this option is not specified, the compiler will prompt if the binary file already exists. Respond with 'Y' to delete it.

E (Display errors only). The compiler normally produces a line-numbered source listing. If this option is selected only error lines (if any) will be displayed.

C (Display code). Output a full listing, including both the source and the interlisted object code.

G (Display globals symbols). Output a symbol table containing only globally-declared symbols (which includes all procedure entry points).

A (Display all symbols). Output a symbol table with both global and local symbols. Each symbol table block will be displayed as the block is exited.

If a binary file is being produced, it will have a transfer address only if an initial origin (e.g. 0A100H:;) is specified as described in Section IX.

If the code option (C) is selected, the object code for each statement is output as it is generated. Since this is a one-pass compiler, occasionally lines like:

```
155C: 7E 00 00
```

are output when the compiler knows that a forward jump is required (for example in an IF or DO-WHILE statement) but doesn't know the addresss yet. In such cases an additional entry is output further down in the listing, when the address is resolved. Parentheses are used to indicate that this entry is a "fixup" to a previous unresolved jump:

```
(155C: 7E 15 90)
```

A symbol table is output only if one of the options A or G is selected. The symbols are alphabetized on the first character only. Along with each symbol is listed the type (BYTE, ADDR, PROC, or LIT), and its value. Appendix C was printed with the G option.

When the compiler has finished executing, it will display the number of errors, followed by the highest memory address used by the symbol table. If the compiler returns to the monitor without displaying these last two items, a fatal error has occurred (see Section XI).

Examples:

```
SPLM                        - Interactive input from keyboard
SPLM,SIZE                   - Source = SIZE.TXT, binary = SIZE.BIN
SPLM,SIZE,+GY               - Source = SIZE.TXT, binary = SIZE.BIN,
                                display globals, delete old binary
SPLM,SIZE,O.SIZE.CMD,+E     - Source = SIZE.TXT, binary = O.SIZE.CMD,
                                display errors only
```

Include Files

The compiler has a built-in include processor, which allows source library files to be brought in during a compile. The syntax is:

```
#INCLUDE <source>
```

where the <source> file name defaults to a .TXT extension and the working drive. The #INCLUDE must start in column 1. The include statement is replaced by the file it includes. When the end of the include file is reached, the compiler switches back to the original file. Included files should not be terminated by an EOF statement, and must not themselves contain #INCLUDE statements (i.e., includes can not be nested).

The source from an included file is normally output to the listing in place of the #INCLUDE statement. However this can be inhibited by the #NOLIST statement:

```
#NOLIST

    source text

#LIST
```

None of the source text between the #NOLIST and the #LIST will be listed, except for any lines in error. Both statements must start in column 1, and neither are output to the listing.

The library files listed in Appendix B are intended to be included at the beginning of an SPL/M program, as needed. All the files have a #NOLIST statement at the beginning, and a #LIST statement at the end, so they won't be listed during every compile.

Printer Considerations

To have the listing output to a printer, precede the SPLM command with a P (see the P command in the FLEX User's Manual). For example,

```
P,SPLM,SIZE
```

would cause the line-numbered source listing for SIZE.TXT (along with any error messages) to be output to the printer.

Each page of the listing starts with a form-feed (0CH) character, which is followed by the top margin, title and finally the source/object listing. The title includes the source file name (without extension), date, and page number and is followed by two blank lines. This title is generated in FLX102.TXT and thus can be changed by the user if desired.

The byte at location 3A2H specifies the top margin, i.e. the number of blank lines from the top of the page to the title. This number can be 0, which will cause the title to be printed on the top line.

The byte at location 3A1H specifies the number of lines to be printed on each page before the formfeed is issued. This count includes the top margin (see above), plus three for the title.

To accomodate narrow-width printers, if the byte at location 039DH = 1 the title and source/object listing is limited to 40 columns (assuming the input source is kept less than 32 characters wide).

Note: printer spooling should not be peformed during a compile, since the compiler reroutes SWI's back to the ROM monitor to handle fatal errors (see Section XI). The SWI vector is restored when the compiler returns to the DOS.

Memory Usage

The main part of the compiler uses RAM from 0380H to 3FFFH. The symbol table starts at location 4000H and can go up to 47FFH. The highest address actually used by the symbol table is displayed at the end of each compile.

The interface routines which link the compiler with the DOS are assembled to reside at 4800H-4FFFH, but they can be easily moved by changing one ORG statement in FLX102.TXT if more room is needed for the symbol table.

The compiler also uses low memory up to location 0EFH. The top of the stack is set to 1FFH on entry but is restored on exit.

XI. ERROR HANDLING

(SSB/FLEX Version 1.2)

When an error is detected, the source line is printed followed by a line containing one or more single-character flags indicating the error(s). The error codes are:

D - Duplicate declaration of the same identifier
O - Origin error (see Section IX for rules)
P - Procedure definition error (Section VII)
S - Syntax error; statement has an illegal construction
U - Undefined identifier

The flags are positioned under the primitive or operator where the error was discovered. For example, in the printout below,

```
0210   TBL(I) = CTR1 ++ CTR2;
****   U              S U
```

TBL and CTR2 are undefined, and there is a syntax error because of the second '+'. When a syntax error is discovered, the remainder of the statement is ignored (up to the next ';'), except that undefined identifiers will continue to be flagged. Also, when undefined identifiers are encountered code is still generated (assuming an address of 0) to allow patching.

The above errors are the only ones which should occur for most users. They are all non-fatal; that is the compile is allowed to proceed.

In addition there are a number of fatal errors which result in the compiler aborting. They are implemented via software interrupts, and result in the ROM monitor (e.g. SWTBUG) being entered.

If the compiler quits and a register dump is displayed, then a fatal error has occurred. The next to the last field of the dump gives the address of the software interrupt, which should be listed on the next page:

0E73 - expression too complex (operator stack overflow)

0E7F - expression too complex (operand stack overflow)

0E89 - expression too complex (expr type stack overflow)

15AB - program too complex (symbol table nesting >64)

1B94 - input line too long (>80 characters)

26A9 - program too complex (fixup jump for IF or DO-WHILE is longer than 512 bytes)

2712 - bad source format (input doesn't end with ODH)

29EF - program too complex (IF chain nest >60)

29FA - identifier too long (>31 characters)

2F83 - out of symbol table memory (as defined by location 0386H)

If any of the above errors occur, return to the DOS via the warm start address, correct the problem and recompile.

If a fatal error occurs that is not listed above, an internal "impossible" compiler error has occurred. Please send the error code plus a listing of the program causing the error to Programma Consultants, using the attached SER (Suspected Error Report) form.

APPENDIX A

SPL/M Compiler Interface Routines

```
                        ********************************************************
                        *                                                      *
                        *        SPL/M COMPILER - INTERFACE ROUTINES           *
                        *      (C) COPYRIGHT 1979 BY THOMAS W. CROSLEY         *
                        *                                                      *
                        *          FLEX 1.0/2.0 COMPILER VERSION 1.2           *
                        *                                                      *
                        *   THIS CODE CONTAINS THE DOS-SPECIFIC ROUTINES       *
                        *   NECESSARY TO INTERFACE THE SPL/M COMPILER          *
                        *   WITH A PARTICULAR OPERATING SYSTEM.                *
                        *                                                      *
                        ********************************************************

                        *
                        * EQUATES FOR FLEX DOS
                        *
0000                    XFC     EQU     0       FUNCTION CODE
0001                    XES     EQU     1       ERROR STATUS
0003                    XUN     EQU     3       UNIT NUMBER
0004                    XFN     EQU     4       FILE NAME
000C                    XEX     EQU     12      EXTENSION
003B                    XSC     EQU     59      SPACE COMP FLAG
0002                    QSO4W   EQU     2       OPEN FOR WRITE
0001                    QSO4R   EQU     1       OPEN FOR READ
0004                    QSCL    EQU     4       CLOSE
000C                    QDEL    EQU     12      DELETE
0003                    FFE     EQU     3       FILE EXISTS
0008                    FEOF    EQU     8       END OF FILE
0001                    TXTEXT  EQU     1       TEXT EXTENSION
0000                    BINEXT  EQU     0       BINARY EXTENSION
0016                    TRNREC  EQU     $16     TRANSFER RECORD
0002                    BINREC  EQU     2       BINARY RECORD
0008                    FNLEN   EQU     8       FILE NAME LEN
B406                    FMS     EQU     $B406
B403                    FMSCLS  EQU     $B403
AD2D                    GETFIL  EQU     $AD2D
AD3F                    RPTERR  EQU     $AD3F
AD03                    WARMS   EQU     $AD03
A080                    IB      EQU     $A080   INPUT LINE BUFFER
AC14                    LINPTR  EQU     $AC14   IB POINTER
AD1B                    INBUFF  EQU     $AD1B
AC18                    CURCHR  EQU     $AC18
AD15                    GETCHR  EQU     $AD15
AD18                    PUTCHR  EQU     $AD18
AD12                    OUTCH2  EQU     $AD12
AD27                    NXTCH   EQU     $AD27
AD33                    SETEXT  EQU     $AD33
AD2A                    RSTRIO  EQU     $AD2A
AD24                    PCRLF   EQU     $AD24
AD39                    OUTDEC  EQU     $AD39
AC0F                    MONTH   EQU     $AC0F
AC0F                    DAY     EQU     $AC0F
AC10                    YEAR    EQU     $AC10
```

```
                        *
                        * EQUATES FOR SWTBUG
E124                    SFE1    EQU     $E124       NON-VECTORED SWI
A012                    SWIJMP  EQU     $A012
                        *
                        * EQUATES TO INTERFACE WITH REST OF COMPILER
0570                    INPOPT  EQU     $570        INPUT OPTION
0571                    PRTOPT  EQU     $571        PRINT OPTION
0572                    OUTOPT  EQU     $572        CODE GENERATION OPTION
0573                    SYMOPT  EQU     $573        SYMBOL TABLE OPTION
3D80                    SBFEND  EQU     $3D80       END OF SOURCE BUF
00C0                    INTORG  EQU     $C0         INITIAL ORIGIN FLAG
003C                    BUFADR  EQU     $3C         CURRENT BUF PTR
003E                    BUFEND  EQU     $3E         END OF BUFFER PTR
                        *
000D                    CR      EQU     $D
0020                    SPACE   EQU     $20
                        *
                        * VECTOR TABLE FOR COMPILER:
                        *
0380                            ORG     $380
                        * COLD START ENTRY POINT
0380 7E 2C 78                   JMP     $2C78
                        *
                        * GETPARMS - JUMP TO USER SUB TO PARSE COMMAND LINE
0383 7E 48 00                   JMP     GPARMS
                        *
                        * HIGH MEMORY - HIGHEST MEM LOC USABLE BY SYMBOL TABLE
0386 47 FF                      FDB     GPARMS-1
                        *
                        * LOADX - ADDRESS OF USER SUB TO TRANSFER BA TO X
0388 00 00                      FDB     0           IF 0, COMPILER WILL GENERATE
                        *
                        * PCRLF - JUMP TO USER ROUTINE TO OUTPUT CRLF
038A 7E AD 24                   JMP     PCRLF
                        *
                        * PUTCHR - JUMP TO USER OUTPUT ROUTINE
038D 7F AD 18                   JMP     PUTCHR
                        *
                        * CASS/DISK READ - JUMP TO USER ROUTINE TO READ SOURCE
0390 7F 49 7D                   JMP     DREAD
                        *
                        * CASS/DISK WRITE - JUMP TO USER ROUTINE TO WRITE OBJECT
0393 7E 4A 65                   JMP     DWRITE
                        *
                        * MULT - ADDRESS OF USER SUB TO MULTIPLY BA BY CONTENTS
                        *       OF BYTES 0,1 -- RESULT IN BA
0396 00 00                      FDB     0           IF 0, COMPILER WILL GENERATE
                        *
                        * DIV - ADDRESS OF USER SUB TO DIVIDE BA BY CONTENTS OF
                        *       BYTES 0,1 -- QUOTIENT IN BA, REMAINDER IN 0,1
0398 00 00                      FDB     0           IF 0, COMPILER WILL GENERATE
                        *
                        *
```

```
                        * LINBUF - ADDRESS OF LINE BUFFER USED BY INBUFF
039A AO 80              LINBUF  FDB     IB
                        *
039C 00                         FCB     0           NOT USED
                        *
                        * NARROW - SET TO 1 IF PRINTER HAS 40 COLUMNS
039D 00                 NARROW  FCB     0
                        *
                        * GETCHR - JUMP TO USER KEYBOARD CHARACTER INPUT ROUTINE
039E 7E AD 15                   JMP     GETCHR
                        *
                        * PLEN - NUMBER OF LINES OUTPUT AFTER FORMFEED
03A1 39                         FCB     57
                        *
                        * TMAR - NUMBER OF BLANK LINES BETWEEN FORMFEED AND TITLE
03A2 02                         FCB     2
                        *
03A3 00                         FCB     0           NOT USED
                        *
                        * LINEIN - JMP TO USER KEYBOARD LINE INPUT ROUTINE
03A4 7E AD 1B                   JMP     INBUFF
                        *
                        * PTITLE - JMP TO USER SUB TO OUTPUT TITLE AT TOP
                        *          OF PAGE
03A7 7E 4B 1F                   JMP     PTITLE
                        *
                        * WRAPUP - JMP TO WRAPUP ROUTINE
03AA 7E 48 44                   JMP     CLOSE
                        *


                        *
                        * NOTE --- THE FOLLOWING CODE IS VECTORED TO FROM LOCATIONS
                        * 380-3AC, AND CAN BE REASSEMBLED ANYWHERE BY CHANGING THE
                        * THE FOLLOWING ORIGIN:
4800                            ORG     $4800
                        *
                        *** NOTE: NEXT 2 INSTRUCTIONS FOR SWTBUG ONLY ***
4800 CE E1 24           GPARMS  LDX     #SFE1       RESTORE NORMAL SWI'S
4803 FF A0 12                   STX     SWIJMP
                        *
4806 7F 05 70                   CLR     INPOPT      CLEAR OPTION FLAGS
4809 7F 05 71                   CLR     PRTOPT
480C 7F 05 72                   CLR     OUTOPT
480F 7F 05 73                   CLR     SYMOPT
4812 7F 4B F3                   CLR     DELOPT
                        *
                        * PARSE THE COMMAND LINE
4815 B6 AC 18                   LDA A   CURCHR
4818 81 0D                      CMP A   #CR
481A 26 09                      BNE     GP10
481C BD AD 2A                   JSR     RSTRIO      INTERACTIVE KEYBOARD OPTION
481F BD 4B 9E                   JSR     ITITLE      OUTPUT TITLE
4822 7E 48 F4                   JMP     GP70
```

```
                        *
                        * SET DEFAULTS FOR DISK INPUT
4825 86 02      GP10    LDA A   #2
4827 B7 05 70           STA A   INPOPT      INPUT FROM DISK
482A B7 05 71           STA A   PRTOPT      SOURCE PRINTOUT
482D 7C 05 72           INC     OUTOPT      PRODUCE BINARY
                        *
4830 7F 4B FE           CLR     INCLP       INCLUDE NEST=0
4833 7F 4B FF           CLR     REOF        READ EOF=FALSE
4836 7F 4C 00           CLR     PAGENO      PAGE NUMBER=0
                        *
                        * PARSE SOURCE FILE NAME
4839 CE 4C 03           LDX     #RFCB
483C BD AD 2D           JSR     GETFIL
483F 24 09              BCC     GP30        BRANCH IF OK
4841 BD AD 3F   ERROR   JSR     RPTERR
4844 BD B4 03   CLOSE   JSR     FMSCLS      CLOSE ALL FILES
4847 7E AD 03           JMP     WARMS
                        *
                        * OPEN SOURCE FILE
484A 86 01      GP30    LDA A   #TXTEXT
484C BD AD 33           JSR     SETEXT      DEFAULT EXT IS .TXT
484F 86 01              LDA A   #QS04R
4851 A7 00              STA A   XFC,X
4853 BD B4 06           JSR     FMS
4856 26 E9              BNE     ERROR
                        *
                        * COPY SOURCE FILE NAME TO BINARY
4858 CE 4C 03           LDX     #RFCB
485B FF 4B F4           STX     XTMP
485E CE 4D 43           LDX     #WFCB
4861 FF 4B F6           STX     XTMP2
4864 BD 49 49           JSR     COPYFN
4867 CE 4D 43           LDX     #WFCB
486A 6F 0C              CLR     XEX,X       CLEAR EXTENSION
486C 6F 0D              CLR     XEX+1,X
486E 6F 0E              CLR     XEX+2,X
                        *
4870 BD AD 27           JSR     NXTCH
4873 81 0D              CMP A   #CR
4875 27 7D              BEQ     GP70        USE DEFAULTS
4877 81 2B              CMP A   #'+
4879 27 16              BEQ     OPTLP       GET OPTIONS
                        *
487B FE AC 14           LDX     LINPTR
487E 09                 DEX
487F FF AC 14           STX     LINPTR      RESET FOR GETFIL
                        *
                        * PARSE BINARY FILE NAME
4882 CE 4D 43           LDX     #WFCB
4885 BD AD 2D           JSR     GETFIL
4888 25 B7              BCS     ERROR
488A BD AD 27           JSR     NXTCH
488D 81 2B              CMP A   #'+
```

```
488F 26 63                BNE    GP70       USE DEFAULTS
                          *
                          * GET OPTIONS (+BYECAG)
4891 BD AD 27  OPTLP      JSR    NXTCH
4894 81 0D                CMP A  #CR
4896 27 5C                BEQ    GP70       ALL DONE
4898 81 42                CMP A  #'B        DON'T PRODUCE BINARY
489A 26 05                BNE    OPT10
489C 7F 05 72             CLR    OUTOPT
489F 20 F0                BRA    OPTLP
48A1 81 59      OPT10     CMP A  #'Y        DELETE OLD BINARY
48A3 26 05                BNE    OPT20
48A5 7C 4B F3             INC    DFLOPT
48A8 20 E7                BRA    OPTLP
48AA 81 45      OPT20     CMP A  #'E        PRINT ERRORS ONLY
48AC 26 07                BNE    OPT30
48AE 86 01                LDA A  #1
48B0 B7 05 71   OPT25     STA A  PRTOPT
48B3 20 DC                BRA    OPTLP
48B5 81 43      OPT30     CMP A  #'C        FULL PRINTOUT WITH CODE
48B7 26 04                BNE    OPT40
48B9 86 03                LDA A  #3
48BB 20 F3                BRA    OPT25
48BD 81 41      OPT40     CMP A  #'A        PRINT ALL SYMBOLS
48BF 26 07                BNE    OPT50
48C1 86 02                LDA A  #2
48C3 B7 05 73   OPT45     STA A  SYMOPT
48C6 20 C9                BRA    OPTLP
48C8 81 47      OPT50     CMP A  #'G        PRINT ONLY GLOBAL SYMBOLS
48CA 26 04                BNE    OPT60
48CC 86 01                LDA A  #1
48CE 20 F3                BRA    OPT45
                          *
48D0 CE 43 D9   OPT60     LDX    #ILLOPT    ILLEGAL OPTION
48D3 BD 4B 6C             JSR    OUTST2
48D6 7F 42 44             JMP    CLOSE
48D9 0D 0A      ILLOPT    FDB    $0D0A
48DB 49                   FCC    'ILLEGAL OPTION SPECIFIED'
48F3 04                   FCB    4
                          *
48F4 7D 05 72   GP70      TST    OUTOPT
48F7 26 01                BNE    GP75
48F9 39                   RTS               NO BINARY
                          *
                          * OPEN BINARY FILE
48FA CE 4D 43   GP75      LDX    #WFCB
48FD 86 00                LDA A  #BINEXT
48FF BD AD 33             JSR    SETEXT     DEFAULT EXT IS .BIN
4902 86 02                LDA A  #QS04W
4904 A7 00                STA A  XFC,X
4906 BD B4 06             JSR    FMS
4909 26 05                BNE    GP80
490B 86 FF                LDA A  #$FF
490D A7 3B                STA A  XSC,X      NO SPACE COMPRESSION
```

```
490F 39                 RTS                 ALL DONE WITH COMMAND LINE
                        *
4910 A6 01      GP80    LDA A   XFS,X      GET ERROR
4912 81 03              CMP A   #EFE        EXISTS ALREADY?
4914 26 30              BNE     ERRORO      SOME OTHER ERROR
4916 7D 4B F3           TST     DELOPT
4919 26 10              BNE     GP90        DELETE OLD BINARY
491B CE 49 61           LDX     #DELMSG
491E BD 4B 6C           JSR     OUTST2
4921 BD AD 15           JSR     GETCHR
4924 81 59              CMP A   #'Y
4926 27 03              BEQ     GP90
4928 7E 48 44           JMP     CLOSE       ABORT
                        *
                        * DELETE OLD BINARY FILE
492B CF 4D 43   GP90    LDX     #WFCB
492E FF 4B F4           STX     XTMP
4931 CE 4E 83           LDX     #IFCB
4934 FF 4B F6           STX     XTMP2
4937 BD 49 49           JSR     COPYFN      USE INCL FCB AS TEMP
493A CF 4E 83           LDX     #IFCB
493D 86 0C              LDA A   #QDEL       DELETE DESTROYS FCB
493F A7 00              STA A   XFC,X
4941 BD B4 06           JSR     FMS
4944 27 B4              BEQ     GP75        NOW GO OPEN IT
4946 7E 43 41   ERRORO  JMP     ERROR
                        *
                        * COPY FILENAME IN FCB(XTMP) TO (XTMP2)
4949 C6 0C      COPYFN  LDA B   #12
494B FF 4B F4   CPLP    LDX     XTMP
494E A6 03              LDA A   XUN,X
4950 08                 INX
4951 FF 4B F4           STX     XTMP
4954 FF 4B F6           LDX     XTMP2
4957 A7 03      CPLP1   STA A   XUN,X
4959 08                 INX
495A FF 4B F6           STX     XTMP2
495D 5A                 DEC B
495E 26 EB              BNE     CPLP
4960 39                 RTS
                        *
4961 0D 0A      DELMSG  FDB     $0D0A
4963 44                 FCC     'DELETE OLD BINARY (Y-N)? '
497C 04                 FCB     4
                        *
                        * READ SOURCE FROM DISK
497D 7D 4B FF   DREAD   TST     REOF
4980 27 05              BEQ     DREAD1
4982 CF 4C 03           LDX     #RFCB
4985 20 63              BRA     ERROR1      TRYING TO READ PAST EOF
                        *
4987 8D 29      DREAD1  BSR     RBFD        READ FIRST BYTE OF SOURCE LINE
4989 7D 4B FF           TST     REOF        END OF FILE?
498C 26 13              BNE     RDONE       YES
```

```
498F 81 23          CMP A  #'#
4990 27 5B          BEQ    INCL       CHECK FOR '#INCLUDE'
4992 8D 0F  DREAD2  BSR    RDLINE     READ REMAINDER OF LINE
4994 C6 3D          LDA B  #SBFEND/256 CHECK FOR BUFFER OVERFLOW
4996 86 80          LDA A  #SBFEND
4998 90 3F          SUB A  BUFEND+1
499A D2 3E          SBC B  BUFEND
499C 26 01          BNE    BH
499E 4D             TST A
499F 22 E6  BH      BHI    DREAD1
49A1 39     RDONE   RTS                READ ENOUGH FOR NOW
            *
49A2 DE 3E  RDLINE  LDX    BUFEND
49A4 A7 00  RL05    STA A  0,X         ASSUMES ONE RBFD BEFORE CALL
49A6 08             INX
49A7 DF 3E          STX    BUFEND
49A9 81 0D          CMP A  #CR
49AB 27 04          BEQ    RL10
49AD 8D 03          BSR    RBFD
49AF 20 F3          BRA    RL05
49B1 39     RL10    RTS
            *
            * READ BYTE FROM DISK
49B2 FF 4B F4 RBFD  STX    XTMP
49B5 CE 4C 03 RBFD0 LDX    #RFCB      DEFAULT IS READ FCB
49B8 7D 4B FE       TST    INCLP
49BB 27 03          BEQ    RBFD1
49BD CE 4E 83       LDX    #IFCB      SWITCH TO INCLUDE FCB
49C0 BD B4 06 RBFD1 JSR    FMS
49C3 27 1E          BEQ    ROK
49C5 A6 01          LDA A  XES,X
49C7 81 08          CMP A  #FEOF      EOF?
49C9 26 1F          BNE    ERROR1
49CB 7D 4B FE       TST    INCLP      YES, CHECK IF IN INCLUDE FILE
49CE 27 0E          BEQ    SEOF
49D0 7F 4B FE       CLR    INCLP      YES, SWITCH BACK TO MAIN
49D3 86 04          LDA A  #QSCL
49D5 A7 00          STA A  XFC,X
49D7 BD B4 06       JSR    FMS        CLOSE INCLUDE FILE
49DA 26 0E          BNE    ERROR1
49DC 20 D7          BRA    RBFD0
49DE 86 01  SEOF    LDA A  #1
49E0 B7 4B FF       STA A  REOF
49E3 4D     ROK     TST A
49E4 27 DA          BEQ    RBFD1      IGNORE NULL CHARS
49E6 FE 4B F4       LDX    XTMP
49E9 39             RTS
49EA 7E 48 41 ERROR1 JMP   ERROR
            *
49ED 8D C3  INCL    BSR    RBFD
49EF 81 49          CMP A  #'I        CHKS FOR JUST '#I'
49F1 27 0B          BEQ    INCL05
49F3 DE 3E          LDX    BUFEND     SOMETHING ELSE, RESTORE
49F5 C6 23          LDA B  #'#
```

```
49F7 E7 00            STA B  0,X
49F9 08               INX
49FA DF 3E            STX    BUFEND
49FC 20 94            BRA    DREAD2     RET WITH 2ND CHAR IN ACCA
49FE 7D 4B FE INCL05  TST    INCLP
4A01 26 48            BNE    INCE       ERROR - NESTED INCLUDE
4A03 8D AD    INCL10  BSR    RBFD
4A05 81 0D            CMP A  #CR
4A07 27 42            BEQ    INCE       ERROR - NO FILENAME
4A09 81 20            CMP A  #SPACE     IGNORE TO NEXT SPACE
4A0B 26 F6            BNE    INCL10
4A0D 8D A3            BSR    RBFD
4A0F 81 0D            CMP A  #CR
4A11 27 38            BEQ    INCE
4A13 FF 03 9A         LDX    LINBUF
4A16 FF AC 14         STX    LINPTR
4A19 A7 00    INCL20  STA A  0,X         COPY FILE SPEC INTO INPUT BUFFER
4A1B 08               INX
4A1C 81 0D            CMP A  #CR
4A1E 27 04            BEQ    INCL30
4A20 8D 90            BSR    RBFD
4A22 20 F5            BRA    INCL20
4A24 CE 4E 83 INCL30  LDX    #IFCB
4A27 BD AD 2D         JSR    GETFIL     PARSE INCLUDE FILE NAME
4A2A 25 14            BCS    INCO
4A2C 86 01            LDA A  #TXTEXT
4A2E BD AD 33         JSR    SETEXT     DEFAULT EXT IS .TXT
4A31 86 01            LDA A  #QS04R     OPEN INCLUDE FILE
4A33 A7 00            STA A  XFC,X
4A35 BD B4 06         JSR    FMS
4A38 26 06            BNE    INCO
4A3A 7C 4B FE         INC    INCLP
4A3D 7F 49 97         JMP    DREAD1
4A40 CE 4A 54 INCO    LDX    #INCMSG
4A43 BD 4B 6C         JSR    OUTST2
4A46 CE 4E 83         LDX    #IFCB
4A49 20 9F            BRA    ERROR1
4A4B CE 4A 54 INCF    LDX    #INCMSG
4A4E BD 4B 6C         JSR    OUTST2
4A51 7E 48 44         JMP    CLOSE
4A54 0D 0A    INCMSG  FDB    $0D0A
4A56 23               FCC    '#INCLUDE ERROR'
4A64 04               FCB    4
                      *
                      * WRITE OBJECT BUFFER TO DISK
4A65 DE 3C    DWRITE  LDX    BUFADR     POINTS TO OBJ BUF
4A67 A6 00            LDA A  0,X        GET RECORD TYPE
4A69 26 04            BNE    W03
4A6B 7F 4B FB         CLR    ISTRT      STRT RECORD INITIALIZATION
4A6E 39               RTS
4A6F 81 FF    W03     CMP A  #$FF
4A71 26 15            BNE    W10
4A73 96 C0            LDA A  INTORG     END RECORD
4A75 27 F7            BEQ    W01
```

```
4A77 86 16          LDA A  #TRNREC   GOTO BLOCK
4A79 BD 4B 0D       JSR    WBTD
4A7C B6 4B FC       LDA A  STRT      TRANSFER ADDR
4A7F BD 4B 0D       JSR    WBTD
4A82 B6 4B FD       LDA A  STRT+1
4A85 7F 4B 0D       JMP    WBTD
                    *
4A88 81 01    W10   CMP A  #1
4A8A 26 F2          BNE    W01
4A8C 08             INX                REGULAR OBJ RECORD (MAX 512 BYTE
4A8D 08             INX
4A8F 08             INX
4A8F FF 4B F8       STX    CODE        SAVE PTR TO BEG OF CODE
4A92 D6 3E    W15   LDA B  BUFEND
4A94 96 3F          LDA A  BUFEND+1
4A96 BC 4B F9       SUB A  CODE+1
4A99 F2 4B F8       SBC B  CODE        BA HAS LENGTH - 1
4A9C 26 5B          BNE    WSEC        IF >128 BYTES, SPLIT UP
4A9F 81 80          CMP A  #$80
4AA0 24 57          BHS    WSEC
4AA2 7D 4B FB       TST    ISTRT
4AA5 26 13          BNE    WBLK
4AA7 81 02          CMP A  #2
4AA9 26 0F          BNE    WBLK
4AAB F6 00          LDA B  0,X
4AAD C1 7E          CMP B  #$7E        DUMMY JUMP ONLY?
4AAF 26 09          BNE    WBLK        DON'T OUTPUT JUST 7E 0000
4AB1 5F             CLR B
4AB2 F1 01          CMP B  1,X
4AB4 26 04          BNE    WBLK
4AB6 E1 02          CMP B  2,X
4AB8 27 3F          BEQ    WRTS
4ABA B7 4B FA WBLK  STA A  COUNT
4ABD 86 02          LDA A  #BINREC     BINARY BLOCK
4ABF 8D 4C          BSR    WBTD
4AC1 DF 3C          LDX    BUFADR
4AC3 A6 01          LDA A  1,X
4AC5 7D 4B FB       TST    ISTRT
4AC8 26 03          BNE    W20
4ACA B7 4B FC       STA A  STRT        REMEMBER INITIAL STRT ADDR
4ACD 8D 3E    W20   BSR    WBTD        WRITE STRT ADDR
4ACF A6 02          LDA A  2,X
4AD1 7D 4B FB       TST    ISTRT
4AD4 26 03          BNE    W30
4AD6 B7 4B FD       STA A  STRT+1
4AD9 8D 32    W30   BSR    WBTD
4ADB 86 01          LDA A  #1
4ADD B7 4B FB       STA A  ISTRT
4AE0 7C 4B FA       INC    COUNT       NORMALIZE LENGTH
4AE3 B6 4B FA       LDA A  COUNT
4AE6 8D 25          BSR    WBTD        WRITE LENGTH
4AE8 FF 4B F8       LDX    CODE
4AEB A6 00    WLOOP LDA A  0,X         WRITE OUT CODE
4AED 8D 1E          BSR    WBTD
```

```
4AFF 08                    INX
4AF0 7A 4B FA              DEC     COUNT
4AF3 26 F6                 BNE     WLOOP
4AF5 FF 4B F8              STX     CODE        SAVE PTR TO NEXT BYTE
4AF8 39           WRTS     RTS
                  *
4AF9 86 7F        WSEC     LDA A   #$7F        WRITE A SECTION (128 BYTES)
4AFB 8D BD                 BSR     WBLK
4AFD DF 3C                 LDX     BUFADR
4AFF E6 01                 LDA B   1,X
4B01 A6 02                 LDA A   2,X
4B03 8B 80                 ADD A   #$80        ADD 128 TO START ADDR
4B05 C9 00                 ADC B   #0
4B07 E7 01                 STA B   1,X
4B09 A7 02                 STA A   2,X
4B0B 20 85                 BRA     W15
                  *
                  * WRITE BYTE TO DISK
4B0D FF 4B F4     WBTD     STX     XTMP
4B10 CE 4D 43              LDX     #WFCB
4B13 BD B4 06              JSR     FMS
4B16 26 04                 BNE     ERROR2
4B18 FE 4B F4              LDX     XTMP
4B1B 39                    RTS
4B1C 7E 48 41     ERROR2   JMP     ERROR
                  *
                  * OUTPUT TITLE AT TOP OF PAGE
4B1F CE 4C 03     PTITLE   LDX     #RFCB
4B22 C6 08                 LDA B   #FNLEN      LENGTH OF FILE NAME
4B24 A6 04        PTTL05   LDA A   XFN,X       GET CHAR OF FN
4B26 26 02                 BNE     PTTL10
4B28 86 20                 LDA A   #SPACE      PAD
4B2A BD AD 18     PTTL10   JSR     PUTCHR
4B2D 08                    INX
4B2E 5A                    DEC B
4B2F 26 F3                 BNE     PTTL05
                  *
4B31 CE 4B BB              LDX     #TITLE0
4B34 BD 4B 5F              JSR     OUTSTR
4B37 B6 03 9D              LDA A   NARROW      40 CHAR PRINTOUT?
4B3A 27 08                 BEQ     PTTL12      NO
4B3C CE 4B C0              LDX     #TITLE2
4B3F BD 4B 5F              JSR     OUTSTR
4B42 20 06                 BRA     PTTL15
4B44 CE 4B C5     PTTL12   LDX     #TITLE3     OUTPUT COMPILER VERSION
4B47 BD 4B 5F              JSR     OUTSTR
4B4A BD 4B 82     PTTL15   JSR     DATE        OUTPUT DATE
4B4D CE 4B EA              LDX     #PAGE
4B50 BD 4B 5F              JSR     OUTSTR
4B53 7C 4C 00              INC     PAGENO
4B56 B6 4C 00              LDA A   PAGENO
4B59 BD 4B 78              JSR     ONEDEC      OUTPUT PAGE NUMBER
4B5C 7E AD 24              JMP     PCRLF
                  *
```

```
                        * SAME AS PSTRNG EXCEPT NO INITIAL CRLF
4B5F A6 00              OUTSTR  LDA A  0,X
4B61 81 04                      CMP A  #4
4B63 27 06                      BEQ    OSRTS
4B65 BD AD 18                   JSR    PUTCHR
4B68 08                         INX
4B69 20 F4                      BRA    OUTSTR
4B6B 39                 OSRTS   RTS
                        *
                        * SAME AS OUTSTR EXCEPT USES OUTCH2
4B6C A6 00              OUTST2  LDA A  0,X
4B6F 81 04                      CMP A  #4
4B70 27 F9                      BEQ    OSRTS
4B72 BD AD 12                   JSR    OUTCH2
4B75 08                         INX
4B76 20 F4                      BRA    OUTST2
                        *
                        * OUTPUT ONE BYTE IN DECIMAL
4B78 B7 4C 02           ONEDEC  STA A  DGT+1
4B7B CE 4C 01                   LDX    #DGT
4B7E 5F                         CLR B                  NO LEADING SPACES
4B7F 7E AD 39                   JMP    OUTDEC
                        *
                        * OUTPUT DATE
4B82 B6 AC 0E           DATE    LDA A  MONTH
4B85 BD 4B 78                   JSR    ONEDEC
4B88 86 2D                      LDA A  #'-
4B8A BD AD 18                   JSR    PUTCHR
4B8D B6 AC 0F                   LDA A  DAY
4B90 BD 4B 78                   JSR    ONEDEC
4B93 86 2D                      LDA A  #'-
4B95 BD AD 18                   JSR    PUTCHR
4B98 B6 AC 10                   LDA A  YEAR
4B9B 7E 4B 78                   JMP    ONEDEC
                        *
                        * TITLE FOR INTERACTIVE USE
4B9E BD AD 24           ITITLE  JSR    PCRLF
4BA1 B6 03 9D                   LDA A  NARROW
4BA4 26 0C                      BNE    ITTL10
4BA6 CE 4B BB                   LDX    #TITLE0
4BA9 BD 4B 5E                   JSR    OUTSTR
4BAC CE 4B BC                   LDX    #TITLE1
4BAF BD 4B 5E                   JSR    OUTSTR
4BB2 CE 4B C5           ITTL10  LDX    #TITLE3
4BB5 BD 4B 5E                   JSR    OUTSTR
4BB8 7E AD 24                   JMP    PCRLF
                        *
4BBB 20                 TITLE0  FCC    '  '
4BBC 20                 TITLE1  FCC    '    '
4BC0 20                 TITLE2  FCC    '    '
4BC4 04                         FCB    4
4BC5 53                 TITLE3  FCC    'SPL/M COMPILER VERSION 1.2  '
4BE9 04                         FCB    4
4BEA 20                 PAGE    FCC    '    PAGE '
```

```
4BF2 04                    FCB     4
                    *
4BF3 00            DELCPT  FCB     0
4BF4 00 00         XTMP    FDB     0
4BF6 00 00         XTMP2   FDB     0
4BF8 00 00         CODE    FDB     0
4BFA 00            COUNT   FCB     0
4BFB 00            ISTRT   FCB     0
4BFC 00 00         STRT    FDB     0
4BFE 00            INCLP   FCB     0
4BFF 00            REOF    FCB     0
4C00 00            PAGENO  FCB     0
4C01 00 00         DGT     FDB     0
                    *
4C03               RFCB    RMB     320
4D43               WFCB    RMB     320
4E83               IFCB    RMB     320
                    *
4FC3               PGEND   EQU     *
                           END
```

NO ERROR(S) DETECTED

SYMBOL TABLE:

BH	499F	BINEXT	0000	BINREC	0002	BUFADR	003C	BUFEND	003E
CLOSE	4844	CODE	4BF8	COPYFN	4949	COUNT	4BFA	CPLP	494B
CPLP1	4957	CR	000D	CURCHR	AC18	DATE	4B82	DAY	AC0F
DELMSG	4961	DELOPT	4BF3	DGT	4C01	DREAD	497D	DREAD1	4987
DREAD2	4992	DWRITE	4A65	EFOF	0008	EFE	0003	ERROR	4841
ERROR0	4946	ERROR1	49EA	ERROR2	4B1C	FMS	B406	FMSCLS	B403
FNLEN	0008	GETCHR	AD15	GETFIL	AD2D	GP10	4825	GP30	434A
GP70	48F4	GP75	48FA	GP80	4910	GP90	492B	GPARMS	4800
IB	A080	IFCB	4E23	ILLOPT	48D9	INBUFF	AD1B	INCE	4A4B
INCL	49ED	INCL05	49FE	INCL10	4A03	INCL20	4A19	INCL30	4A24
INCLP	4BFE	INCMSG	4A54	INCO	4A40	INPOPT	0570	INTORG	00C0
ISTRT	4BFB	ITITLF	4B9E	ITTL10	4BB2	LINBUF	039A	LINPTR	AC14
MONTH	AC0E	NARROW	039D	NXTCH	AD27	ONEDEC	4B78	OPT10	48A1
OPT20	48AA	OPT25	48B0	OPT30	48B5	OPT40	48BD	OPT45	48C3
OPT50	48C8	OPT60	48D0	OPTLP	4891	OSRTS	4B6B	OUTCH2	AD12
OUTDEC	AD39	OUTOPT	0572	OUTST2	4B6C	OUTSTR	4B5F	PAGE	4BEA
PAGENO	4C00	PCRLF	AD24	PGEND	4FC3	PRTOPT	0571	PTITLE	4B1F
PTTL05	4B24	PTTL10	4B2A	PTTL12	4B44	PTTL15	4B4A	PUTCHR	AD18
QDEL	000C	QSCL	0004	QSO4R	0001	QSO4W	0002	RBFD	49B2
RBFDO	49B5	RBFD1	49C0	RDLINE	49A2	RDONE	49A1	REOF	4BFF
RFCB	4C03	RL05	49A4	RL10	49B1	ROK	49F3	RPTERR	AD3F
RSTRIO	AD2A	SBFEND	3D80	SEOF	49DE	SETEXT	AD33	SFE1	F124
SPACE	0020	STRT	4BFC	SWIJMP	A012	SYMOPT	0573	TITLE0	4BBB
TITLF1	4BBC	TITLE2	4BC0	TITLE3	4BC5	TRNREC	0016	TXTEXT	0001
W01	4A6E	W03	4A6F	W10	4A88	W15	4A92	W20	4ACD
W30	4AD9	WARMS	AD03	WBLK	4ABA	WBTD	4B0D	WFCB	4D43
WLOOP	4AEB	WRTS	4AF8	WSEC	4AF9	XFS	0001	XEX	000C
XFC	0000	XFN	0004	XSC	003B	XTMP	4BF4	XTMP2	4BF6
XUN	0003	YEAR	AC10						

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

APPENDIX B

SPL/M DOS Library Routines

```
#NOLIST
/*  SPLM LIBRARY 'SPLM.LIB' —
    DOS INTERFACE ROUTINES

    FLEX VERSION 1.0  6-9-79  */

/*  THESE ROUTINES CAN BE USED BY AN
    SPLM PROGRAM TO INTERFACE WITH
    THE DOS.  PARAMETERS NORMALLY
    PASSED IN REGISTERS ARE PLACED
    IN GLOBAL VARIABLES INSTEAD.

    SEE THE FLEX 2.0 "ADVANCED PRO-
    GRAMMERS GUIDE" FOR A DETAILED
    DESCRIPTION OF EACH OF THE
    ROUTINES.

    THE VERSION NUMBER OF THE PROGRAM
    MUST BE DECLARED AS A SYMBOLIC
    CONSTANT BEFORE INCLUDING THIS
    FILE.  THE STARTING ADDRESS AND ANY
    GLOBAL VARIABLES NOT ON PAGE 0 (SUCH
    AS ARRAYS) SHOULD ALSO BE DECLARED
    BEFORE THE LIBRARY INCLUDES, E.G.

    0A100H:;
    DCL VERSION LIT '1';

    0A840H: DCL RECB (320) BYTE;
    #INCLUDE SPLM.LIB
    #INCLUDE SPLMREAD.LIB

    VARIABLES DECLARED AFTER THE INCLUDES
    WILL BE PLACED ON PAGE 0 UNLESS
    PRECEDED BY AN ORIGIN.        */

/* GENERATE VERSION NUMBER */
GEN(/*BRA 1*/201H,VERSION);

/* OVERLAY FOR PART OF DOS MEMORY MAP */
0A080H: DCL LINBUF (128) BYTE;
0AC02H: DCL FOLCHR BYTE;
0AC0EH: DCL SMONTH BYTE, SDAY BYTE, SYEAR BYTE;
0AC11H: DCL LASTTERM BYTE;
0AC14H: DCL LINPTR ADDR;
0AC18H: DCL CURCHR BYTE, PREVCHR BYTE;

DCL TRUE LIT '0FFH';
DCL FALSE LIT '0';
DCL CRLF LIT '0D0AH';

/* SYMBOLIC CONSTANTS FOR DISK IO */
DCL XFC LIT '0'; /* FCB OVERLAY */
DCL XES LIT '1';
```

```
DCL XUN LIT '3';
DCL XFN LIT '4';
DCL XEX LIT '12';
DCL XFS LIT '15';
DCL XNC LIT '59';
DCL QSRW LIT '0'; /* FUNCTION DEFS */
DCL QSO4R LIT '1';
DCL QSO4W LIT '2';
DCL QSO4U LIT '3';
DCL QSCLS LIT '4';
DCL QSREW LIT '5';
DCL EEOF LIT '8'; /* ERROR STATUS */
DCL DXBIN LIT '0'; /* DEFAULT EXTENSIONS */
DCL DXTXT LIT '1';
DCL DXCMD LIT '2';
DCL DXSYS LIT '4';
DCL DXBAK LIT '5';
DCL DXOUT LIT '11';

WARMS:PROC;
        GEN(/*JMP*/7EH,0AD03H);
END;

10H:DCL CHAR BYTE;
/* READ ONE BYTE INTO CHAR */
GETCHR:PROC;
        CALL /*GETCHR*/0AD15H;
        GEN(/*STAA*/097H,.CHAR);
END;
/* WRITE ONE BYTE FROM CHAR */
PUTCHR:PROC;
        GEN(/*LDAA*/096H,.CHAR);
        CALL /*PUTCHR*/0AD18H;
END;
/* OUTPUT A SPACE */
SPACE:PROC;
        GEN(/*LDAA*/086H,' ');
        CALL /*PUTCHR*/0AD18H;
END;

DCL INBUFF LIT '0AD1BH';
DCL MSGA ADDR;
/* OUTPUT STRING WHOSE ADDRESS
   IS IN MSGA */
PSTRNG:PROC;
        GEN(/*LDX*/0DEH,.MSGA);
        CALL /*PSTRNG*/0AD1EH;
END;

DCL ERROR BYTE;
/* CLASSIFY CHAR; ERROR = TRUE
   IF NOT ALPHANUMERIC */
CLASS:PROC;
        ERROR = 0FFH;
```

```
        GEN(/*LDAA*/96H,.CHAR);
        CALL /*CLASS*/0AD21H;
        GEN(/*BCC*/24H,1); RETURN;
        ERROR = 0;
END;
DCL PCRLF LIT '0AD24H';
/* GET NEXT BUFFER CHARACTER
   INTO CHAR */
NXTCH:PROC;
        CALL /*NXTCH*/0AD27H;
        GEN(/*STAA*/97H,.CHAR);
END;
DCL RSTRIO LIT '0AD2AH';

DCL FCBA ADDR;
/* GET FILE SPEC INTO FCB WHOSE
   ADDRESS IS IN FCBA.  NORMALLY
   ONLY CALLED BY LIBRARY ROUTINES
   RDOPEN AND WTOPEN */
GETFIL:PROC;
        ERROR = 0FFH;
        GEN(/*LDX*/0DEH,.FCBA);
        CALL /*GETFIL*/0AD2DH;
        GEN(/*BCC*/24H,1); RETURN;
        ERROR = 0;
END;
DCL LOAD LIT '0AD30H';
DCL DEFEXT BYTE;
/* SET DEFAULT EXTENSION
   CONTAINED IN DEFEXT */
SETEXT:PROC;
        GEN(/*LDAA*/96H,.DEFEXT);
        GEN(/*LDX*/0DEH,.FCBA);
        CALL /*SETEXT*/0AD33H;
END;

DCL DGTA ADDR, LDSPC BYTE;
/* OUTPUT DECIMAL NUMBER WHOSE
   ADDRESS IS IN DGTA.  LEADING
   SPACES WILL BE PRINTED IF
   LDSPC = TRUE */
OUTDEC:PROC;
        GEN(/*LDAB*/0D6H,.LDSPC);
        GEN(/*LDX*/0DEH,.DGTA);
        CALL /*OUTDEC*/0AD39H;
END;
/* OUTPUT HEX BYTE WHOSE
   ADDRESS IS IN DGTA */
OUTHEX:PROC;
        GEN(/*LDX*/0DEH,.DGTA);
        CALL /*OUTHEX*/0AD3CH;
END;

/* REPORT DOS ERRORS.  NORMALLY
```

```
   ONLY CALLED FROM DISK I/O
   LIBRARY ROUTINES */
RPTERR:PROC;
        GEN(/*LDX*/0DEH,.FCBA);
        CALL /*RPTERR*/0AD3FH;
END;

DCL NUM ADDR, ANYDGTS BYTE;
/* GET HEX NUMBER INTO NUM.
   ERROR SET TRUE IF NOT HEX.
   DGTS SET <> 0 IF ANY DIGITS
   FOUND.  */
GETHEX:PROC;
        NUM=0; ERROR=0FFH; ANYDGTS=0;
        CALL /*GETHEX*/0AD42H;
        GEN(/*BCC*/24H,1); RETURN;
        ERROR=0;
        GEN(/*STX*/0DFH,.NUM);
        GEN(/*STAB*/0D7H,.ANYDGTS);
END;
/* OUTPUT 2 HEX BYTES WHOSE
   ADDRESS IS IN DGTA */
OUTADR:PROC;
        GEN(/*LDX*/0DEH,.DGTA);
        CALL /*OUTADR*/0AD45H;
END;
/* INPUT DECIMAL NUMBER INTO NUM.
   ERROR SET IF INVALID NUMBER.
   DGTS SET <> 0 IF ANY DIGITS
   FOUND. */
INDEC:PROC;
        NUM=0; ERROR=0FFH; ANYDGTS=0;
        CALL /*INDEC*/0AD48H;
        GEN(/*BCC*/24H,1); RETURN;
        ERROR=0;
        GEN(/*STX*/0DFH,.NUM);
        GEN(/*STAB*/0D7H,.ANYDGTS);
END;

DOCMND:PROC;
        CALL /*DOCMND*/0AD4EH;
        GEN(/*STAB*/0D7H,.ERROR);
END;
FMS:PROC;
        /* SET ERROR = 0FFH WITHOUT
           DESTROYING CHAR IN ACCA */
        ERROR = 0; ERROR = ERROR-1;
        GEN(/*LDX*/0DEH,.FCBA);
        CALL /*FMS*/0B406H;
        GEN(/*BEQ*/27H,1); RETURN;
        ERROR = 0;    /* ACCA STILL HAS CHAR */
END;
DCL FMSCLS LIT '0B403H';
#LIST
```

```
#NOLIST
/*  SPLM LIBRARY 'SPLMREAD.LIB' --
          READ ROUTINES

     FLEX VERSION 1.0  6-9-79  */

/*  THESE ROUTINES CAN BE USED BY AN
    SPLM PROGRAM TO READ A SEQUENTIAL
    FILE.  A FILE CONTROL BLOCK NAMED
    'RFCB' MUST BE DECLARED BEFORE
    THE LIBRARY INCLUDE, E.G.:

    0A840H: DCL RFCB (320) BYTE;
    #INCLUDE SPLM.LIB
    #INCLUDE SPLMREAD.LIB        */


/*  RDCLOSE - CLOSE A FILE PREVIOUSLY
    OPENED FOR READING */

RDCLOSE:PROC;
        RFCB(XFC) = QSCLS;
        FCBA = .RFCB;
        CALL FMS;
        IF ERROR THEN DO;
                CALL RPTERR;
                CALL WARMS;
        END;
END;

/*  RDER - HANDLE FATAL READ ERRORS */

RDER:PROC;
        FCBA = .RFCB;
        CALL RPTERR;
        CALL RDCLOSE;
        CALL WARMS;
END;

/*  RDOPEN - OPEN A FILE FOR READING.
    ON ENTRY, (GLOBAL) DEFEXT MUST
    CONTAIN THE DEFAULT EXTENSION
    TYPE - SEE 'SPLM.LIB' FOR
    SYMBOLIC CONSTANTS TO USE.
    SPACE COMPRESSION IS ALWAYS
    INHIBITED BY DEFAULT */

RDOPEN:PROC;
        FCBA = .RFCB;
        CALL GETFIL;
        IF ERROR THEN DO;
                CALL RPTERR;
                CALL WARMS;
        END;
```

```
        RFCB(XFC) = QS04R;
        CALL SETFXT;  /* DEFFXT MUST BE SET UP */
        CALL FMS;
        IF ERROR THEN DO;
                CALL RPTERR;
                CALL WARMS;
                END;
        /* INHIBIT SPACE COMP */
        RFCB(XNC) = TRUE;
END;

/*  RBFT - READ ONE BYTE FROM DISK
    INTO (GLOBAL) CHAR.
    ON EXIT, REOF = TRUE IF END OF
    FILE, ELSE REOF = FALSE */

DCL REOF BYTE;
RBFD:PROC;
        REOF = TRUE;
        RFCB(XFC) = QSRW;
        FCBA = .RFCB;
        CALL FMS;
        GEN(/*STAA*/97H,.CHAR);
        IF ERROR THEN DO;
                IF RFCB(XES) = FEOF THEN RETURN;
                ELSE CALL RDER;
        END;
        REOF = FALSE;
END;

/*  RBFDF - READ ONE BYTE FROM DISK
    INTO (GLOBAL) CHAR.  END OF
    FILE HANDLED AS FATAL ERROR */

RBFDF:PROC;
        CALL RBFD;
        IF REOF THEN CALL RDER;
END;
#LIST
```

```
#NOLIST
/*  SPLM LIBRARY 'SPLMWRIT.LIB' —
          WRITE ROUTINES

     FLEX VERSION 1.0  6-9-79  */

/*  THESE ROUTINES CAN BE USED BY AN
    SPLM PROGRAM TO WRITE A SEQUENTIAL
    FILE.  A FILE CONTROL BLOCK NAMED
    'WFCB' MUST BE DECLARED BEFORE
    THE LIBRARY INCLUDES, E.G.:

    100H: DCL RFCB (320) BYTE,
          DCL WFCB (320) BYTE;
    #INCLUDE SPLM.LIB
    #INCLUDE SPLMREAD.LIB
    #INCLUDE SPLMWRIT.LIB        */


/*  WTCLOSE - CLOSE A FILE PREVIOUSLY
    OPENED FOR WRITING */

WTCLOSE:PROC;
        WFCB(XFC) = QSCLS;
        FCBA = .WFCB;
        CALL FMS;
        IF ERROR THEN DO;
                CALL RPTERR;
                CALL WARMS;
        END;
END;

/*  WTER - HANDLE FATAL READ ERRORS */

WTER:PROC;
        FCBA = .WFCB;
        CALL RPTERR;
        CALL WTCLOSE;
        CALL WARMS;
END;

/*  WTOPEN - OPEN A FILE FOR WRITING.
    ON ENTRY, (GLOBAL) DEFEXT MUST
    CONTAIN THE DEFAULT EXTENSION
    TYPE - SEE 'SPLM.LIB' FOR
    SYMBOLIC CONSTANTS TO USE.
    SPACE COMPRESSION IS ALWAYS
    INHIBITED BY DEFAULT */

WTOPEN:PROC;
        FCBA = .WFCB;
        CALL GETFIL;
        IF ERROR THEN DO;
                CALL RPTERR;
```

```
                CALL WARMS;
        END;
        WFCB(XFC) = QSO4W;
        CALL SETEXT;  /* DEFEXT MUST BE SET UP */
        CALL FMS;
        IF ERROR THEN DO;
                CALL RPTERR;
                CALL WARMS;
                END;
        /* INHIBIT SPACE COMP */
        WFCB(XNC) = TRUE;
END;

/*  WBTD - WRITE ONE BYTE FROM (GLOBAL)
    CHAR TO DISK.     */

WBTD:PROC;
        WFCB(XFC) = QSRW;
        FCBA = .WFCB;
        GEN(/*LDAA*/96H,.CHAR);
        CALL FMS;
        IF ERROR THEN CALL WTER;
END;
#LIST
```

APPENDIX C

"Size" Program (SPL/M Source)

```
0001  /* SIZE — DISPLAYS SECTOR COUNT,  */
0002  /* LENGTH IN DECIMAL AND HEX,     */
0003  /* NUMBER OF LINES (CR'S), PLUS   */
0004  /* CHECKSUM AND CREATION DATE OF  */
0005  /* A FILE.                        */
0006  /*                                */
0007  /*        FLEX VERSION 1.0        */
0008  /*            6-11-79             */
0009
0010  0A100H:;
0011  DCL VERSION LIT '1';
0012
0013  0A840H:DCL RFCB (320) BYTE;
0014
0015  /* #INCLUDE SPLM.LIB        — LIBRARIES INCLUDED HERE
0016     #INCLUDE SPLMREAD.LIB    */
0322
0323  DATE:PROC;      /* OUTPUT DATE AS  MM-DD-YY  */
0324          DCL MONTH LIT '25', DAY LIT '26', YEAR LIT '27';
0325          DCL DGT ADDR;
0326          LDSPC = FALSE;
0327          IF RFCB(MONTH) < 10 THEN CALL SPACE;
0328          DGTA = .DGT;
0329          DGT = RFCB(MONTH); CALL OUTDEC;
0330          CHAR = '-'; CALL PUTCHR;
0331          DGT = RFCB(DAY); CALL OUTDEC;
0332          CHAR = '-'; CALL PUTCHR;
0333          DGT = RFCB(YEAR); CALL OUTDEC;
0334          IF RFCB(DAY) < 10 THEN CALL SPACE;
0335          CALL SPACE;
0336  END;
0337
0338  ASIZE:PROC;     /* OUTPUT SIZE AND CHECKSUM INFO FOR A FILE */
0339          DCL BYTE$CTR ADDR, LINE$CTR ADDR, CHKSUM BYTE;
0340          DCL TBYTE$CTR ADDR, FLAG BYTE;
0341          DCL XSIZ LIT '21'; /* LOC OF SECTOR SIZE IN FCB */
0342          DCL CR LIT '0DH';
0343
0344          BYTE$CTR = 0; LINE$CTR = 0; FLAG = FALSE; CHKSUM = 0;
0345          CALL RBFD;
0346          DO WHILE NOT REOF;
0347                  IF FLAG AND (CHAR <> 0) THEN FLAG = FALSE;
0348                  IF NOT FLAG AND (CHAR = 0) THEN DO;
0349                          FLAG = TRUE;
0350                          /* MARK LAST NON-ZERO BYTE */
0351                          TBYTE$CTR = BYTE$CTR;
0352                  END;
0353                  CHKSUM = CHKSUM + CHAR;
0354                  BYTE$CTR = BYTE$CTR + 1;
0355                  IF CHAR = CR THEN LINE$CTR = LINE$CTR + 1;
0356                  CALL RBFD;
0357          END;
```

```
0358            IF FLAG THEN        /* STRING OF NULLS AT END */
0359                    BYTE$CTR = TBYTE$CTR;
0360
0361            LDSPC = TRUE;
0362            DGTA = .RFCB+XSIZ; CALL OUTDEC; /* SECTOR SIZE */
0363            CALL SPACE;
036
0365            DGTA = .BYTE$CTR; CALL OUTDEC;  /* BYTE COUNT */
0366            CALL SPACE; CALL SPACE;
0367
0368            CALL OUTADR;                    /* IN HEX */
0369            CALL SPACE;
0370
0371            DGTA = .LINE$CTR; CALL OUTDEC;  /* LINE COUNT */
0372            CALL SPACE; CALL SPACE;
0373
037             DGTA = .CHKSUM; CALL OUTHEX;    /* CHECKSUM */
0375    END;
0376
0377    /* MAIN */
0378    DCL HEADER DATA (' DATE      NS   DEC   HEX LINES  CS',
0379                     CRLF,CRLF,4);
0380
0381    DFFEXT = DXTXT;
0382    CALL RDOPEN;
0383
038     MSGA = .HEADER; CALL PSTRNG;
0385    CALL DATE;
0386    CALL ASIZE;
0387
0388    CALL RDCLOSE;
0389    CALL WARMS;
0390

0391    LVL 00

001C    ANYDGTS BYTE
A2A8    ASIZE PROC
AC18    CURCHR BYTE
0D0A    CRLF LIT
0010    CHAR BYTE
A12     CLASS PROC
0000    DXBIN LIT
0001    DXTXT LIT
0002    DXCMD LIT
0004    DXSYS LIT
0005    DXBAK LIT
000B    DXOUT LIT
0016    DFFEXT BYTE
0017    DGTA ADDR
A19E    DOCMNT PROC
A253    DATE PROC
```

```
AC02  EOLCHR BYTE
0008  EFOF LIT
0013  ERROR BYTE
0000  FALSE LIT
0014  FCBA ADDR
A1A4  FNS PROC
B403  FNSCLS LIT
A10A  GETCHR PROC
A138  GETFIL PROC
A164  GETHEX PROC
A366  HEADER BYTE
AD1E  INBUFF LIT
A184  INDEC PROC
A080  LINBUF BYTE
AC11  LASTTERM BYTE
AC14  LINPTR ADDR
AD30  LOAD LIT
0019  LDSPC BYTE
0011  MSGA ADDR
A132  NXTCH PROC
001A  NUM ADDR
A150  OUTDEC PROC
A158  OUTHEX PROC
A17E  OUTADR PROC
AC19  PREVCHR BYTE
A110  PUTCHR PROC
A11C  PSTRNG PROC
AD24  PCRLF LIT
0000  QSRW LIT
0001  QSO4R LIT
0002  QSO4W LIT
0003  QSO4U LIT
0004  QSCLS LIT
0005  QSREW LIT
A840  RFCB BYTE
AD2A  RSTRIO LIT
A15E  RPTERR PROC
A1B6  RDCLOSE PROC
A1D  RDER PROC
A1E1  RDOPEN PROC
001D  RFOF BYTE
A216  RDFD PROC
A244  RDFDE PROC
AC0E  SMONTH BYTE
AC0F  SDAY BYTE
AC10  SYEAR BYTE
A116  SPACE PROC
A148  SETEXT PROC
00FF  TRUE LIT
0001  VERSION LIT
A106  WARMS PROC
0000  XFC LIT
```

```
0001  XFS LIT
0003 /XUN LIT
0004  XFN LIT
000C  XFX LIT
000F  XFS LIT
003B  XNC LIT

0391  EOF

****  NO ERRORS

HIGH ADDR USED: 44D6
```

APPENDIX E

Grammar For SPL/M

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

Grammar for SPL/M V1.1

```
<program> ::= <init> <main> EOF

<init> ::= <istmt list> ¦ <origin> ; <istmt list>

<istmt list> ::= <istmt> ¦ <istmt list> <istmt> ¦ NIL

<istmt> ::= <decl stmt> ; ¦ <proc def> ; ¦ <gen stmt> ;

<origin> ::= <number>:

<proc def> ::= <proc head> <stmt list> END

<proc head> ::= <identifier>: PROCEDURE ;
              ¦ <identifier>: PROC ;
              ¦ <origin> <proc head>

<main> ::= <stmt list> ¦ <origin> <stmt list>

<stmt list> ::= <stmt> ¦ <stmt list> <stmt> ¦ NIL

<stmt> ::= <basic stmt> ¦ <if stmt>

<basic stmt> ::= <assignment> ;
               ¦ <group> ;
               ¦ <call stmt> ;
               ¦ RETURN ;
               ¦ BREAK ;
               ¦ <decl stmt> ;
               ¦ <gen stmt> ;

<if stmt> ::= <if clause> <stmt>
            ¦ <if clause> <basic stmt> ELSE <stmt>

<if clause> ::= IF <expr> THEN

<group> ::= <group head> <stmt list> END

<group head> ::= DO ;
               ¦ DO WHILE <expr> ;

<call stmt> ::= CALL <identifier> ¦ CALL <number>
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

APPENDIX D

SPL/M Reserved Words

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

SPL/M Reserved Words

ADDR	LIT
ADDRESS	LITERALLY
AND	* LOW
** BASED	* MEM
BREAK	* MEMA
** BY	** MINUS
BYTE	MOD
CALL	** MONITOR
DATA	NOT
DCL	OR
DECLARE	** PLUS
DO	PROC
ELSE	PROCEDURE
END	RETURN
EOF	THEN
GEN	** TO
GENERATE	WHILE
* HIGH	XOR
IF	

* – Reserved word in Version 1 only

** – Reserved word in future versions; illegal in Version 1

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

```
<decl stmt> ::= DECLARE <decl element>
              ¦ DCL <decl element>
              ¦ <decl stmt> , <decl element>
              ¦ <origin> <decl stmt>

<decl element> ::= <identifier> <type>
              ¦ <identifier> ( <number> ) <type>
              ¦ <identifier> DATA <data list>
              ¦ <identifier> LITERALLY '<number>'
              ¦ <identifier> LIT '<number>'

<type> ::= BYTE ¦ ADDRESS ¦ ADDR

<data list> ::= <data head> <constant> )

<data head> ::= ( ¦ <data head> <constant> ,

<gen stmt> ::= GENERATE <data list>
              ¦ GEN <data list>

<assignment> ::= <variable> = <expr>

<expr> ::= <logical factor>
              ¦ <expr> OR <logical factor>
              ¦ <expr> XOR <logical factor>

<logical factor> ::= <logical secondary>
              ¦ <logical factor> AND <logical secondary>

<logical secondary> ::= <logical primary>
              ¦ NOT <logical primary>

<logical primary> ::= <arith expr>
              ¦ <arith expr> <relation> <arith expr>

<relation> ::= = ¦ < ¦ > ¦ <> ¦ <= ¦ >=

<arith expr> ::= <term>
              ¦ <arith expr> + <term>
              ¦ <arith expr> - <term>

<term> ::= <secondary>
              ¦ <term> * <secondary>
              ¦ <term> / <secondary>
              ¦ <term> MOD <secondary>
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

```
<secondary> ::= <primary>
                ¦ - <primary>

<primary> ::= <constant>
              ¦ <variable>
              ¦ ( <expr> )
              ¦ HIGH ( <expr> )
              ¦ LOW ( <expr> )

<variable> ::= <identifier>
               ¦ <identifier> ( <expr> )
               ¦ MEM ( <expr> )
               ¦ MEMA ( <expr> )

<constant> ::= <number> ¦ '<string>' ¦ .<identifer>

<identifier> ::= <letter>
                 ¦ <identifier> <dec digit>
                 ¦ <identifier> <letter>
                 ¦ <identifier> $

<letter> ::= A ¦ B ¦ C ... ¦ Z

<number> ::= <dec number> ¦ <hex number> H

<dec number> ::= <dec digit>
                 ¦ <dec num> <dec digit>
                 ¦ <dec num> $

<hex number> ::= <dec digit>
                 ¦ <hex num> <hex digit>
                 ¦ <hex num> $

<dec digit> ::= 0 ¦ 1 ¦ 2 ... ¦ 9

<hex digit> ::= <dec digit> ¦ A ¦ B ¦ C ¦ D ¦ E ¦ F

<string> ::= <str element> ¦ <string> <str element>

<str element> ::= <ASCII char> ¦ ''
```